Dipartimento di Scienze e Tecnologie - CdL in Informatica

UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

PROGETTO DI LABORATORIO DI RETI DI CALCOLATORI

BUONAUGURIO CRISTINA 0124/1372
D'ANGELO GIOVANNI 0124/1369

TRACCIA 2 - BLOCKEXPLORER

20/02/2019

SOMMARIO

DESCRIZIONE

Si vuole realizzare un sistema per la visualizzazione e l'analisi delle transazioni memorizzate in una blockchain. Il sistema è basato sulla gestione di una blockchain, ovvero una sequenza di blocchi in cui ogni blocco contiene una transazione.
Il sistema si compone di un nodo NodoN, un server BlockServer ed uno o più client BlockClient. Il NodoN gestisce una copia della blockchain. Il BlockServer si connette al NodoN da cui riceve ogni nuovo blocco che viene aggiunto alla blockchain. Il BlockServer estrae le transazioni dai blocchi e salva una copia locale. Il BlockClient si connette al BlockServer per ricevere informazioni riguardo le transazioni.

DESCRIZIONE DETTAGLIATA

I protagonisti sono NodoN, BlockServer, BlockClient.

NODON

Il NodoN gestisce una copia della blockchain, una sequenza di blocchi in cui ogni blocco contiene una transazione. Un blocco contiene: il numero di blocco progressivo, tempo di attesa random, transazione (IP_PORTA_MITT:AMMONTARE:IP_PORTA_DEST:NUMERO_RANDOM).
Una blockchain inizia con un blocco genesi. Il NodoN memorizza una transazione in un blocco, attendere un tempo random (in [5,15] secondi) che viene memorizzato in un blocco ed infine inserisce il blocco in testa alla blockchain.
Quando si connette un BlockServer, il NodoN inizia ad inviare tutti i blocchi già memorizzati, a partire dal numero di blocco indicato dal server, e tutti quelli che via via inserisce nella blockchain fino a che il BlockServer non si disconnettere.

BLOCKSERVER

Il BlockServer si connette al NodoN e riceve i blocchi, a partire dall'ultimo blocco già ricevuto. Per ogni blocco, estrae le informazioni relative alla transazione e le memorizza localmente. Il BlockServer, inoltre, fornisce informazioni e statistiche ai BlockClient. In particolare può: inviare le ultime N transazioni; una transazione specifica (identificata dal numero progressivo del blocco che la contiene);[1] la somma del valore di tutte le transazioni; visualizzare tutte le transazioni in cui è coinvolto uno specifico indirizzo (identificato da una coppia IP:PORTA).

BLOCKCLIENT

L'utente, utilizzando il BlockClient, può connettersi al BlockServer e visualizzare le informazioni relative alla blockchain. In particolare può visualizzare le ultime N transazioni memorizzate nella blockchain, visualizzare i dettagli relativi ad una specifica transazione, richiedere il valore complessivamente scambiato con le transazioni memorizzate fino al momento della richiesta ed infine visualizzare tutte le transazioni in cui è coinvolto uno specifico indirizzo (IP:PORTA).

---

[1] Al momento della progettazione si è ritenuto più opportuno che il Client inviasse l'id della transazione come chiave in quanto quest'ultimo non ha *motivo di conoscere* la struttura dati che incapsula queste informazioni.

DESCRIZIONE DELL'ARCHITETTURA

PREMESSE

Prima di discutere dell'architettura, è bene aggiungere alcune considerazioni, fatte al momento della progettazione, riguardanti il NodoN.
Il NodoN deve gestire sia l'aggiornamento della blockchain in locale, attraverso l'inserimento da tastiera di nuovi blocchi, sia in rete con il BlockServer connesso.
La gestione della lettura da tastiera da parte del NodoN renderebbe l'esecuzione confusa durante la fase di simulazione e ai fini di questo progetto. Perciò si è deciso di inserire "un'assistente" denominato CreaBlocchi che si occuperà di leggere da tastiera i nuovi blocchi e avere un'esclusiva connessione con il NodoN a cui invierà di volta in volta i blocchi da aggiungere alla blockchain. Così facendo, il NodoN dovrà gestire la ricezione di nuovi blocchi da parte di CreaBlocchi e l'emissione degli stessi al BlockServer eventualmente connesso.
Nelle pagine seguenti, mostreremo la descrizione in fasi dell'architettura.
A meno di eventi da descrivere in cui riteniamo sia necessario mostrare la partecipazione di CreaBlocchi, quest'ultimo verrà omesso. Ricordiamo, comunque, che CreaBlocchi risulta **essenziale** per l'esecuzione dell'intero sistema.

GENERICHE

Il progetto è stato realizzato secondo un modello client/server utilizzando il linguaggio C e le API fornite dallo standard POSIX. Il sistema è costituto da un Client che invia richieste a un Server(BlockServer). Il BlockServer è connesso a sua volta con un altro Server (NodoN) che gli fornisce gli aggiornamenti alla sua copia della blockchain.
Il BlockServer gestisce la connessione fra il Client e NodoN attraverso la **select()**: quando si connette al NodoN viene eseguita una *system call*, una **fork()**, e il processo figlio gestisce l'inserimento di un nuovo blocco attraverso una **pipe()** con il padre, permettendo l'effettivo aggiornamento locale della blockchain. Al momento di una richiesta di connessione del Client, viene eseguita un'altra **fork()** e il processo figlio si occuperà di inviare al Client le informazioni richieste da quest'ultimo.
Il NodoN, inizialmente composto dal solo Nodo Genesi, attende inizialmente la connessione con il processo CreaBlocchi. Avvenuta la connessione, entra in **select()** gestendo la ricezione di nuovi blocchi da parte di CreaBlocchi e restando in ascolto di un BlockServer. Il NodoN dopo aver ricevuto un nuovo blocco e averlo inserito nella sua copia locale, controlla che sia connesso il BlockServer. In quel caso, il NodoN invia il blocco al BlockServer.
Alla prima richiesta di connessione da parte del BlockServer, il NodoN attraverso una **fork()** crea un processo figlio che si occupa di inviare al BlockServer tutta la blockchain memorizzata fino a quel momento a partire dall'ultimo blocco che possiede quest'ultimo. [2]

[2]Al fine della simulazione, si suppone che alla prima connessione il BlockServer abbia 0 blocchi già memorizzati.

FASI

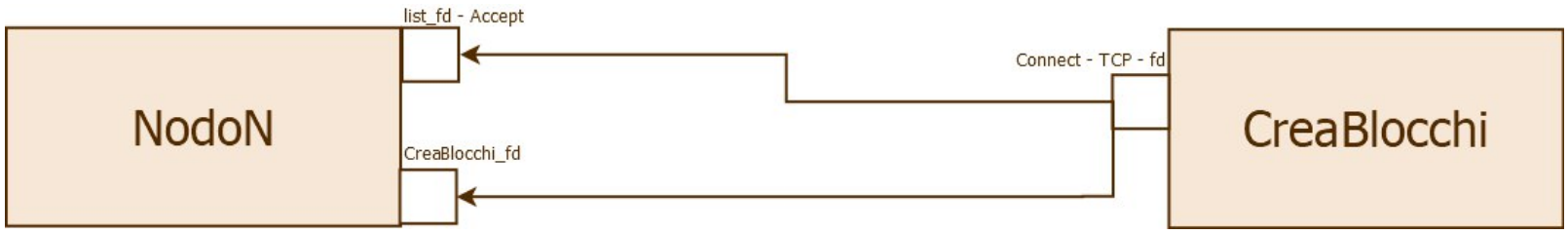
Le fasi che andremo analizzare saranno le seguenti:

- **Fase 0**: CreaBlocchi si connette al NodoN.
- **Fase 1**: BlockServer si connette al NodoN per la prima volta.
- **Fase 2**: Inserimento nuovo blocco.
- **Fase 3**: Un Client si connette al BlockServer. [3]

1. **Fase 3.1**:Il Client chiede di visualizzare le ultime N transazioni.
2. **Fase 3.2**:Il Client chiede di visualizzare il valore complessivamente scambiato.
3. **Fase 3.3**:Il Client chiede di visualizzare una specifica transazione.
4. **Fase 3.4**:Il Client chiede di visualizzare tutte le transazioni in cui è coinvolto un IP:Porta.
5. **Fase 3 D**:Il Client si disconnette.

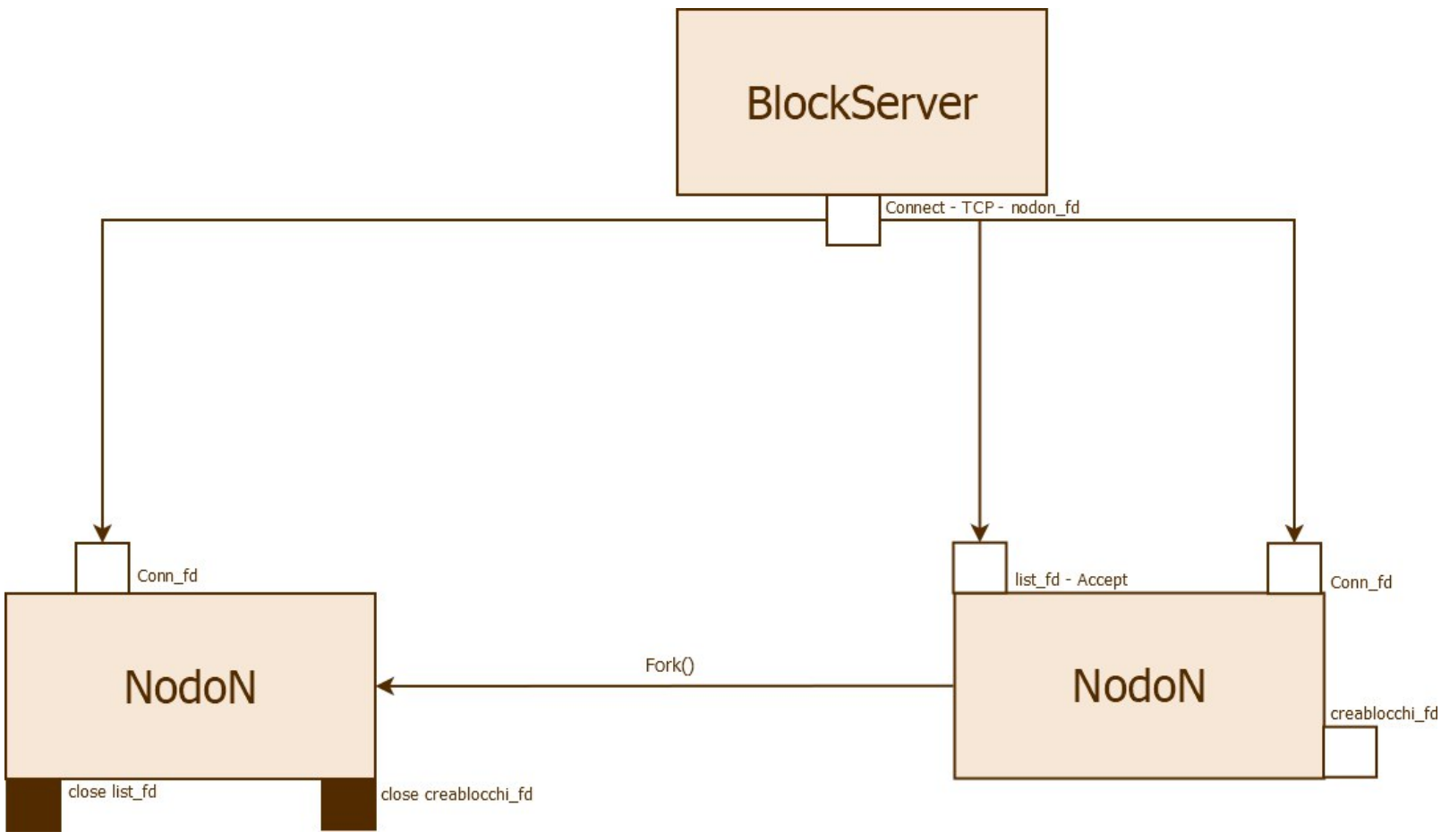
[3] Le fasi seguenti alla fase 3 verranno visualizzate soltanto durante la descrizione dei protocolli.

FASE 0



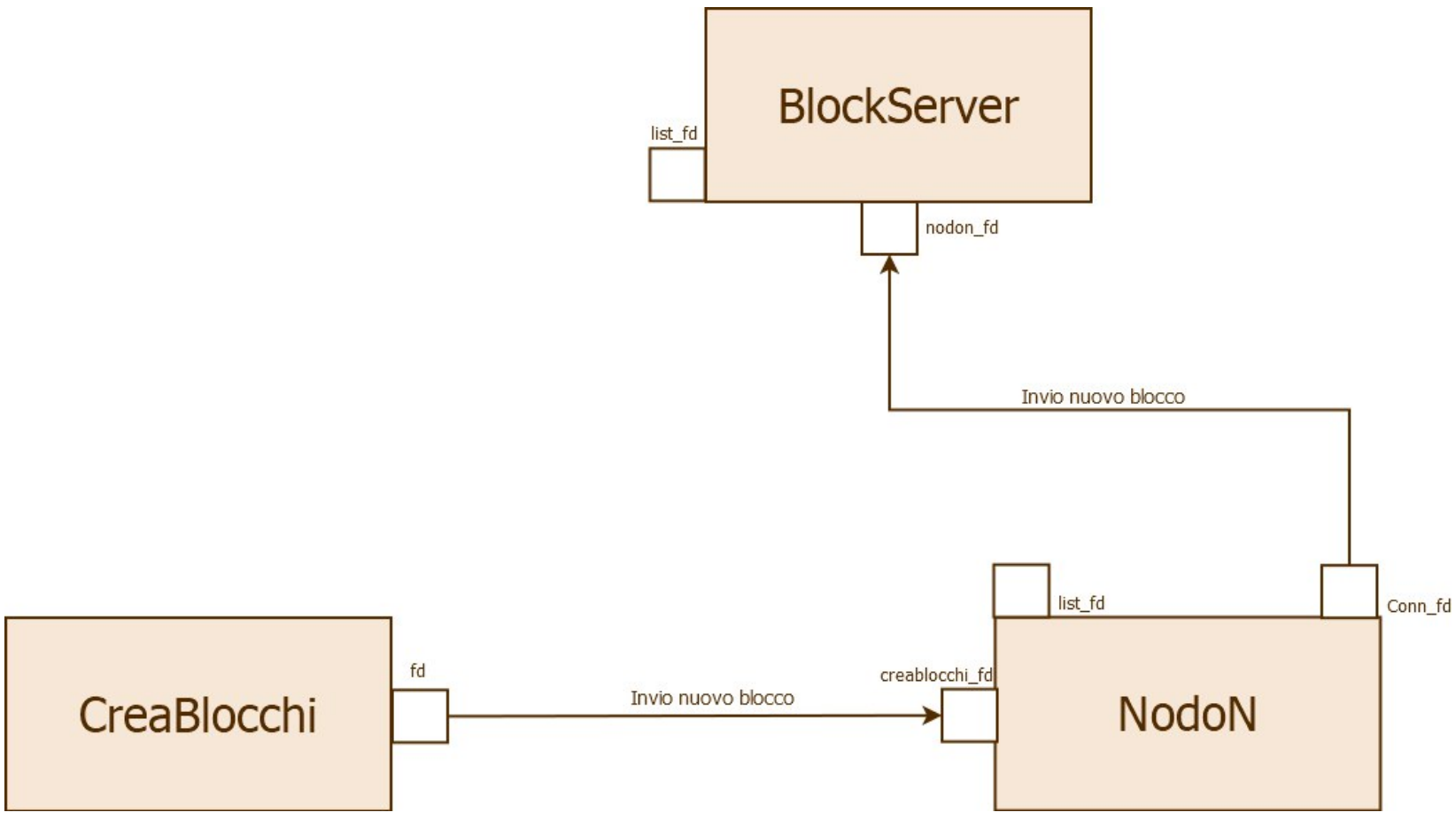
Nella fase 0 si effettua prima l'esecuzione di NodoN che attende la richiesta di connessione di CreaBlocchi. Questa connessione sarà attiva fino a che NodoN sarà in esecuzione. Nel caso in cui CreaBlocchi dovesse disconnettersi, NodoN sarebbe disconnesso a sua volta e viceversa.

FASE 1



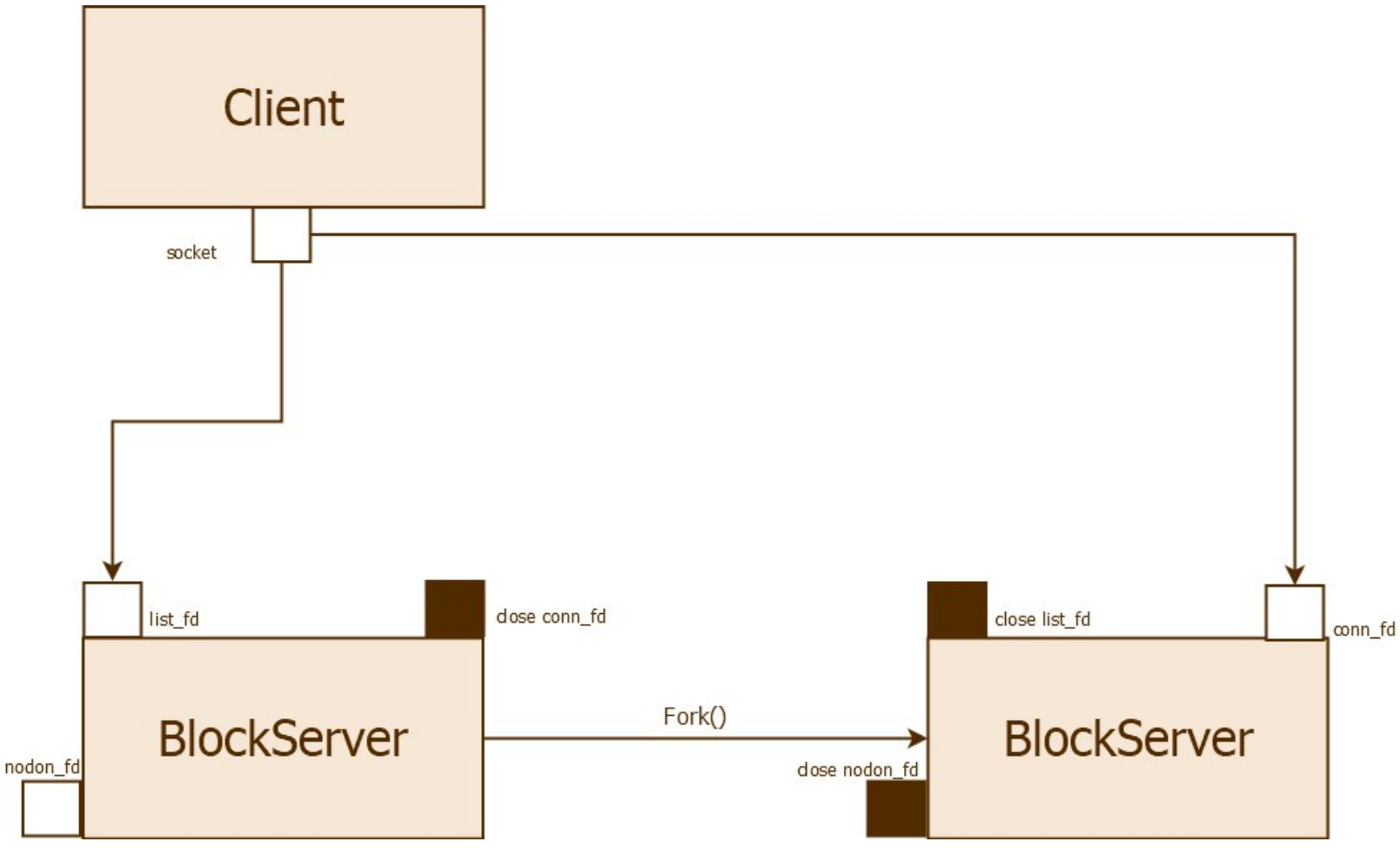
All'avvio il BlockServer manda una richiesta di connessione al NodoN. Il NodoN che in questo momento è in una Select(), procede alla creazione di un processo figlio tramite una Fork() in cui chiude sia il descrittore per la Listen() sia quello per CreaBlocchi. Il processo figlio si occupa di inviare tutti i blocchi per l'aggiornamento della blockchain in BlockServer. La connessione, nel processo padre, non viene chiusa per poter inviare eventualmente i nuovi blocchi che NodoN riceve da CreaBlocchi.

FASE 2



Questa fase non è fondamentale nell'esecuzione del sistema. Se ci troviamo dopo la Fase 1, quindi il BlockServer è connesso al NodoN, ogni qual volta che NodoN riceve un nuovo blocco da CreaBlocchi automaticamente NodoN invia lo stesso a BlockServer, dopo aver aggiornato la sua blockchain.

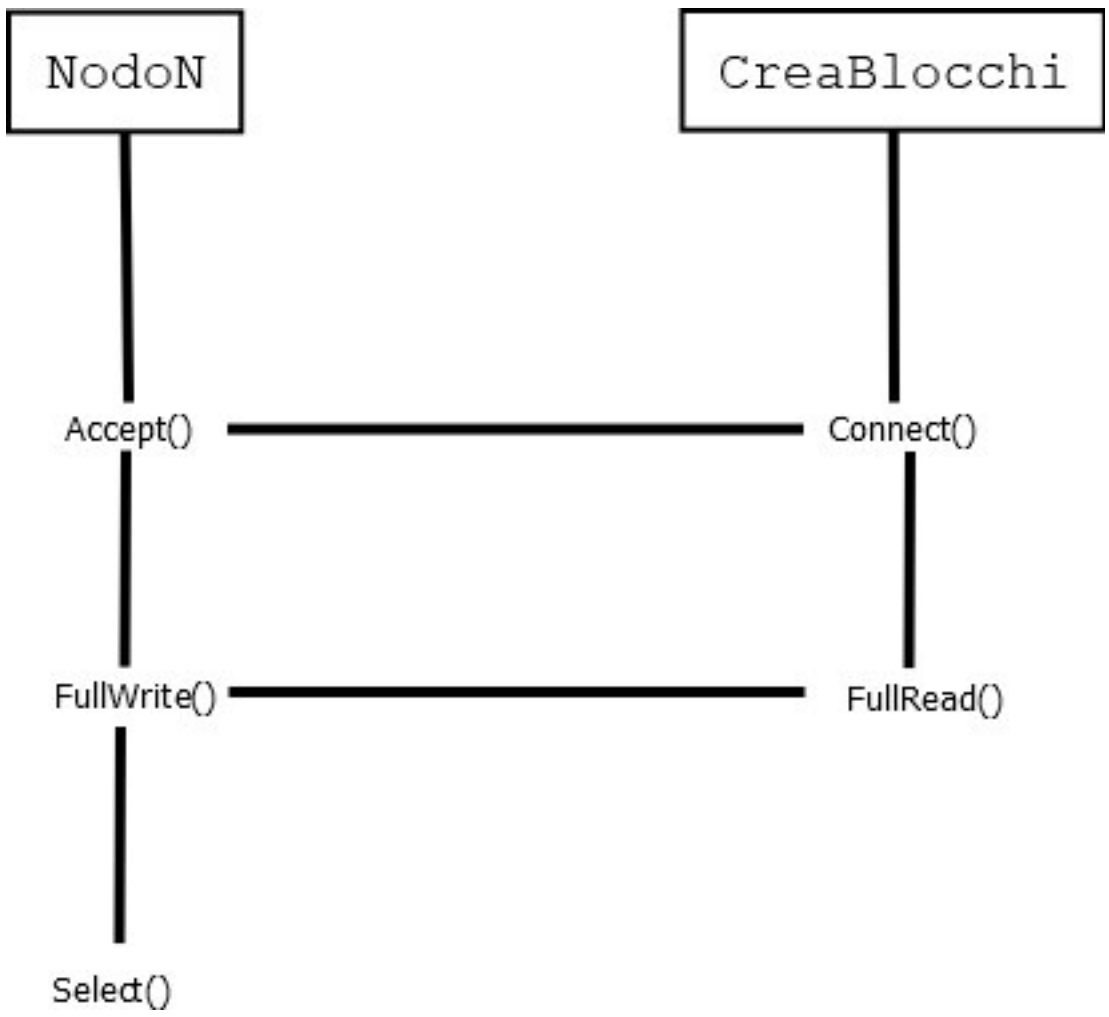
FASE 3



In quest'ultima fase, il Client manda una richiesta di connessione al BlockServer il quale accetta e crea un processo figlio tramite una fork() che processerà le richieste del Client. Il BlockServer resta sempre in comunicazione anche con il NodoN e riceve, eventualmente, nuovi blocchi.

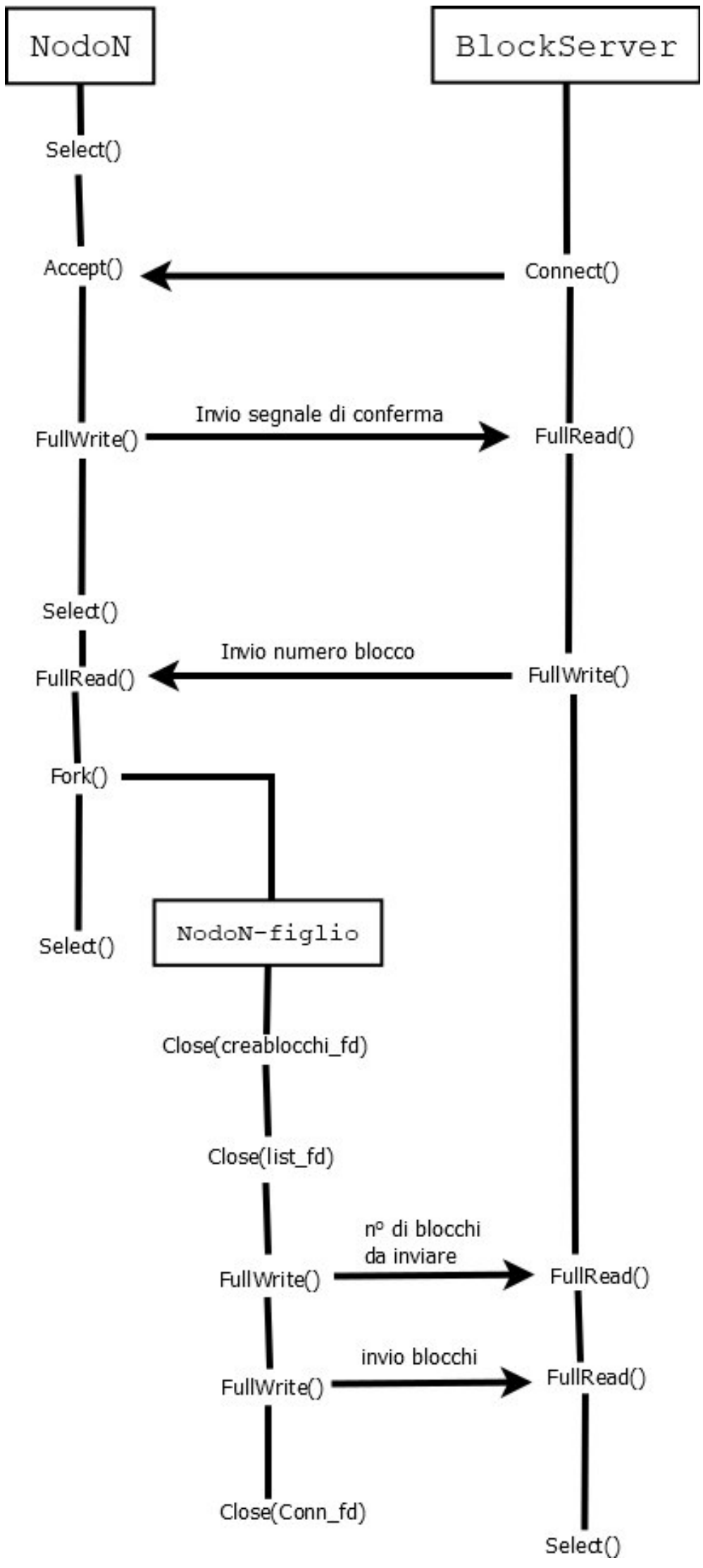
DESCRIZIONE DEI PROTOCOLLI

FASE 0



Il NodoN si blocca sulla **Accept()** attendendo che CreaBlocchi faccia una richiesta di connessione. Appena riceve la richiesta, invia un segnale di successo a CreaBlocchi dopodiché il NodoN si mette in **Select()**, in attesa di una nuova connessione da parte di BlockServer e/o della ricezione di un nuovo blocco. CreaBlocchi, nel frattempo, attende un input da tastiera per l'inserimento di un nuovo blocco, che invierà al NodoN.

FASE 1

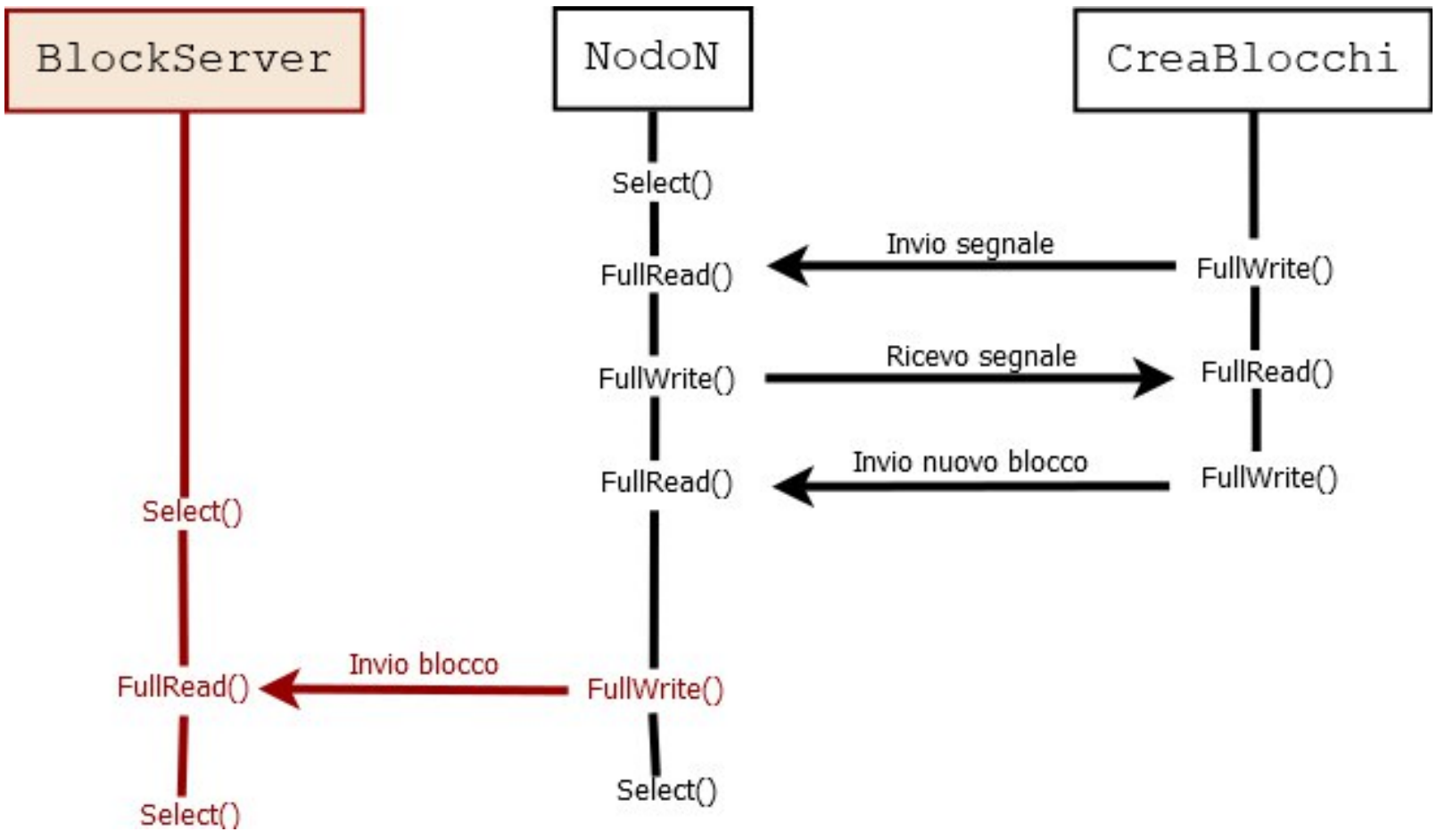

NodoN
BlockServer
Select()
Accept()
Connect()
Invio segnale di conferma
FullWrite()
FullRead()
Select()
Invio numero blocco
FullRead()
FullWrite()
Fork()
Select()
NodoN-figlio
Close(creablocchi_fd)
Close(list_fd)
nº di blocchi
da inviare
FullWrite()
FullRead()
invio blocchi
FullWrite()
FullRead()
Close(Conn_fd)
Select()

Il NodoN si trova in **Select()**, riceve una richiesta di connessione da parte di BlockServer e dopo averla accettata, invia un segnale di conferma. Avvenuta la connessione, ritorna in **Select()** aggiungendo il descrittore di BlockServer tra i descrittori da monitorare.
BlockServer invia l'id dell'ultimo numero di blocco ricevuto[4] a NodoN.
Si crea un processo figlio tramite una **Fork()** che facendo accesso alla sua blockchain, calcola quanti blocchi inviare e poi li invia, in un ciclo iterativo.
Quando saranno inviati tutti i blocchi, il NodoN figlio chiude la connessione con il BlockServer mentre rimane attiva quella col processo padre.[5]

[4] Essendo la prima connessione, sarà zero. Questa FullRead() ha una doppia funzione: permette di gestire una disconnessione improvvisa di BlockServer o leggere l'ultimo id del blocco ricevuto.

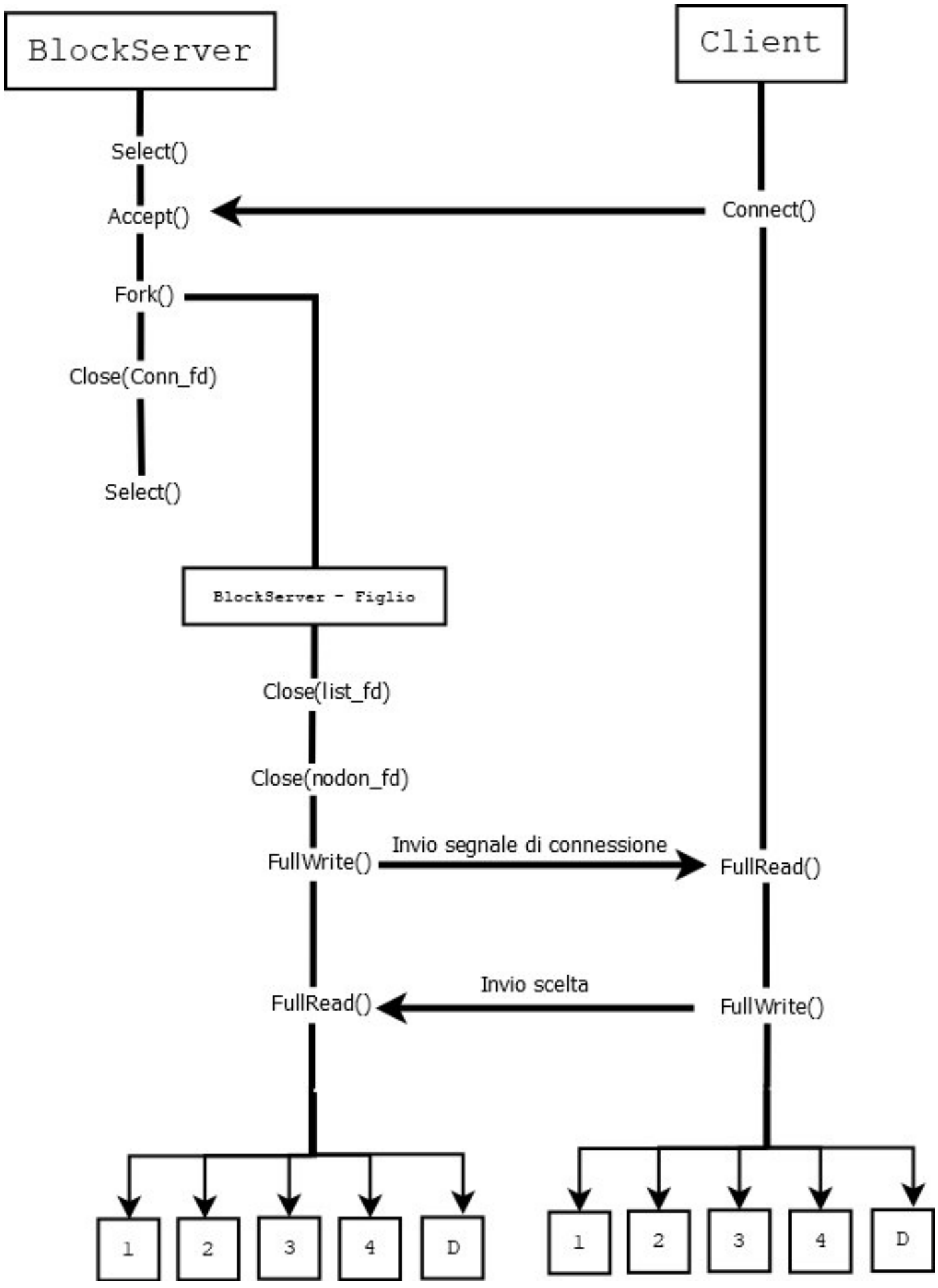
[5] La select() continuerà a monitorare anche il descrittore riservato a BlockServer.

FASE 2



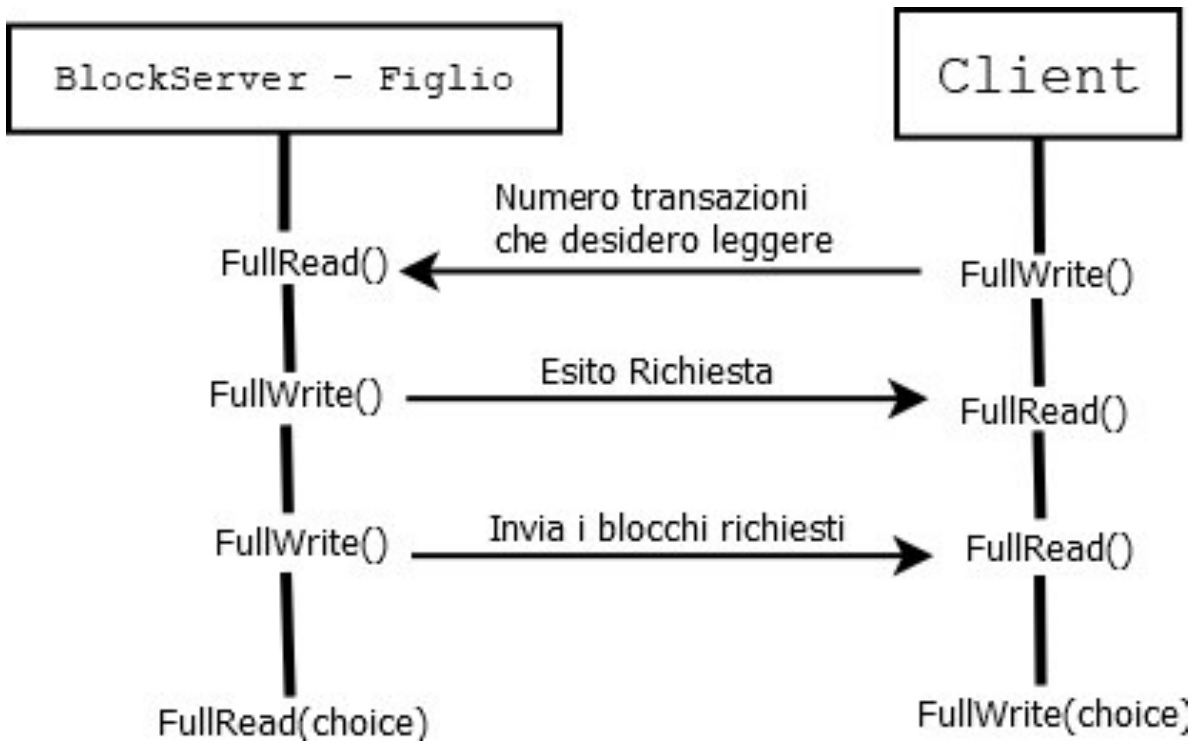
Quando CreaBlocchi è pronto per inviare un nuovo blocco, invia prima un segnale, una volta ricevuta la conferma, invia il blocco al NodoN che lo inserisce in testa alla blockchain. Prima di tornare in Select() controlla se BlockServer è connesso, in tal caso invia il nuovo blocco al BlockServer() il quale poi torna in Select(). CreaBlocchi una volta inviato un blocco, si rimette in ascolto da tastiera per un eventuale nuovo blocco.

FASE 3



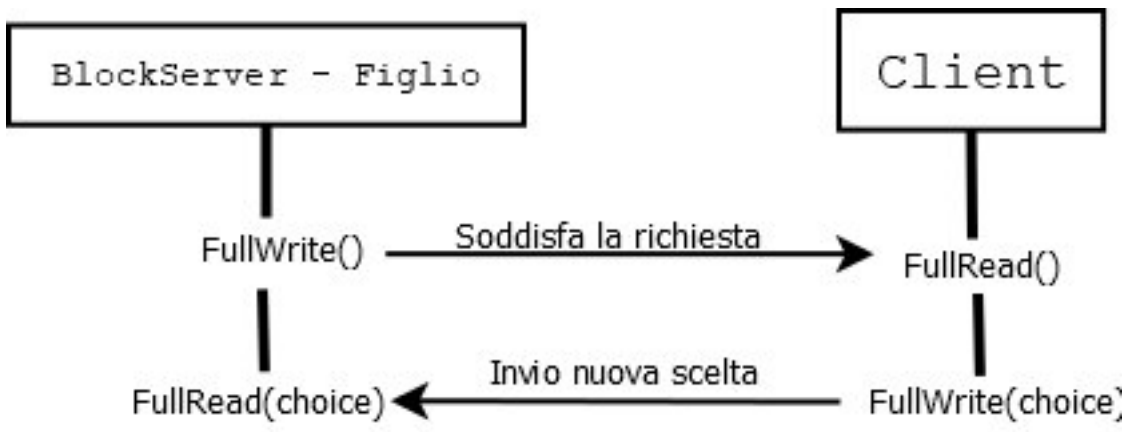
Lo schema illustra la procedura di connessione fra un Client con il BlockServer. Dopo essersi connesso, il Client aspetta che il server sia pronto, dopodiché invierà la scelta presa in input dall'utente. In base al valore della scelta (choice) verranno eseguite 4 protocolli differenti, più il caso D (default) in cui l'utente ha deciso di disconnettersi da BlockServer.

FASE 3.1



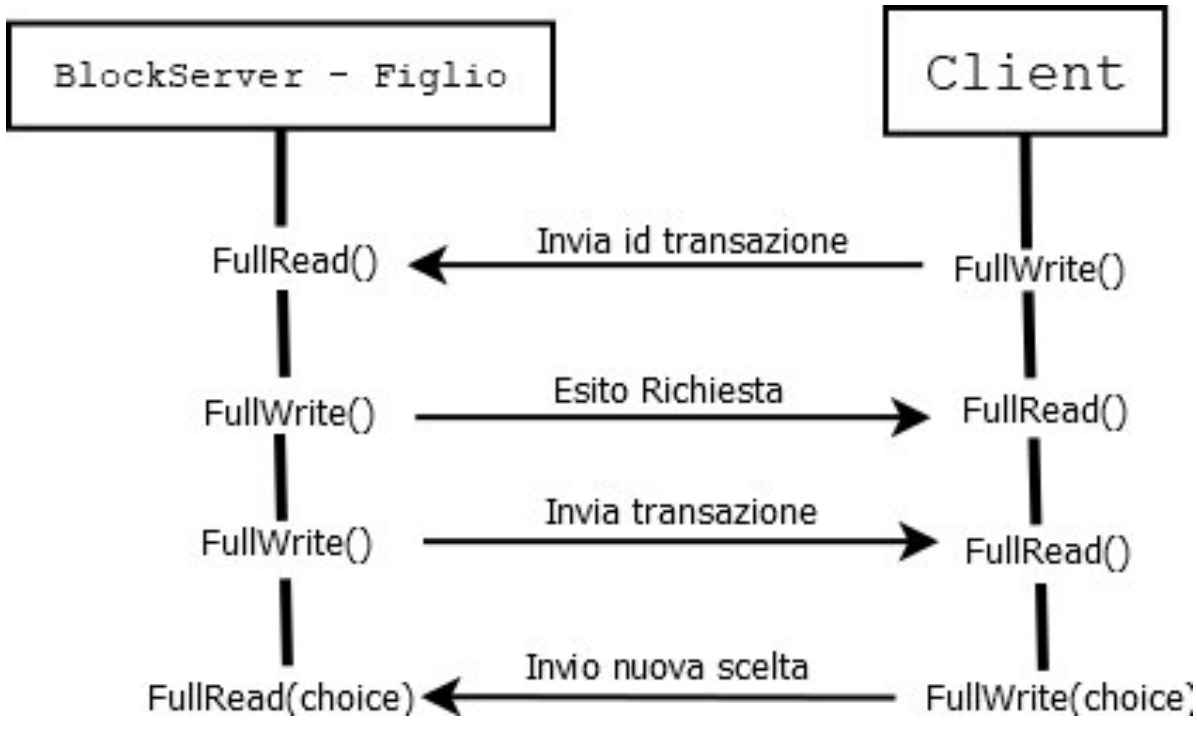
Il Client richiede di visualizzare le ultime N transazioni. Il BlockServer, ricevuto il numero, si accerta di poter soddisfare la richiesta, inviando poi l'esito al Client.
Se l'esito va a buon fine, il BlockServer procede (in un ciclo iterativo) ad inviare il numero di transazioni richiesto, in caso contrario, ritorna ad attendere una nuova scelta del Client.

FASE 3.2



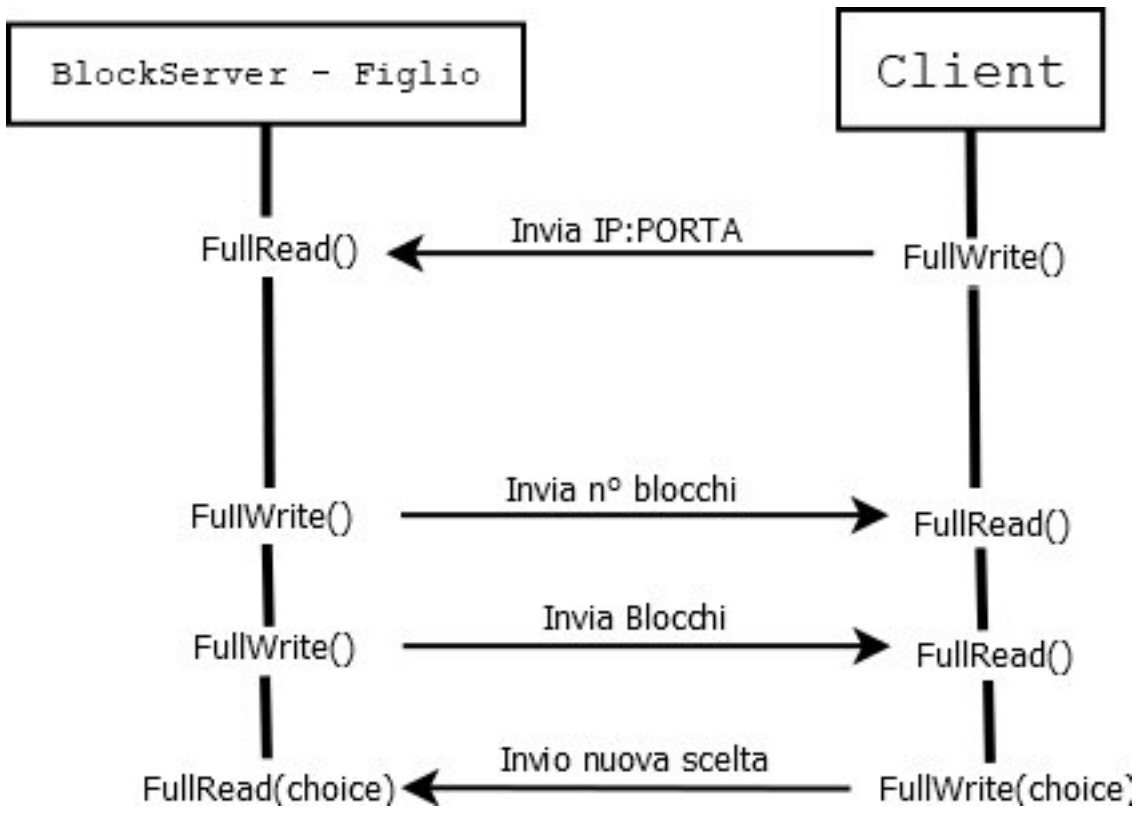
Il Client chiede al BlockServer il valore complessivamente scambiato fra tutte le transazioni. Il BlockServer soddisfa la richiesta e ritorna in FullRead() in attesa di una nuova richiesta da parte del Client.

FASE 3.3



Il Client richiede di visualizzare una specifica transazione, il BlockServer si accerta di avere quella specifica transazione e invia l'esito della sua ricerca al Client. Se l'esito è andato a buon fine, il BlockServer invia la transazione e dopo si rimette in ascolto per una nuova richiesta da parte del Client.

FASE 3.4



Il Client invia al BlockServer un pacchetto indirizzo di tipo sockaddr_in con l'IP e la Porta presi in input. Il BlockServer fa una ricerca e invia così l'esito col numero di transazioni trovate, se ne ha trovato almeno uno, invia in un ciclo iterativo tutte le transazioni.

FASE D (DEFAULT)

Inserendo da tastiera qualsiasi altro carattere, il Client procede alla disconnessione dal BlockServer, il quale non ricevendo più nulla dalla FullRead() chiude il suo descrittore. Questa procedura avviene anche se il Client viene disconnesso.

IMPLEMENTAZIONE

LISTA FILE

Wrapper.h
Header.h
Function.c
NodoN.c
CreaBlocchi.c
BlockServer.c
Client.c

WRAPPER.H

```c
#ifndef WRAPPER_H
#define WRAPPER_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <errno.h>

int Socket(int family, int type, int protocol)
{
int n;
if ( (n = socket(family, type, protocol)) < 0) {
perror("socket");
exit(1);
}
return(n);
}

int Connect(int socket,struct sockaddr *address, socklen_t address_len)
{
    int n;
  if ((n = connect(socket, address, address_len)) < 0) {
    perror("connect");
    exit(1);
  }
  return n;
}

int Bind(int socket, const struct sockaddr *address, socklen_t addrlen)
{
    int n;
    if ((n= bind(socket, address, addrlen)) < 0)
        {perror("bind");
            exit(1);
        }
        return n;
}

int Listen(int socket, int coda)
{
    int n;
     if ( listen(socket, coda) < 0 ) {
    perror("listen");
    exit(1);
  }
  return n;
}

int Accept(int socket, struct sockaddr *clientaddr, socklen_t *addr_dim)
{
    int n;
    if ( ( n = accept(socket, clientaddr, addr_dim) ) < 0 ) {
      perror("accept");
      exit(1);
    }
    return n;
}

ssize_t FullWrite(int fd, const void *buf, size_t count)
```

```c
  {
  size_t nleft;
  ssize_t nwritten;
  nleft = count;
  while (nleft > 0) // La scrittura è ripetuta nel ciclo fino all'esaurimento del numero di byte richiesti
  {
    if ((nwritten = write(fd, buf, nleft)) < 0)
      {
        if (errno == EINTR) // In caso di errore si controlla se è dovuto a un'interruzione della chiamata di sistema dovuta ad un segnale
          continue; // Ripeto l'accesso
        else
          return(nwritten); // Altrimenti l'errore viene ritornato al programma chiamante
      }
    nleft -= nwritten; // Setta la variabile left per la funzione write
    buf +=nwritten; // Setta il puntatore
  }
  return (nleft);
  }

ssize_t FullRead (int fd, void *buf, size_t count)
  {
  size_t nleft;
  ssize_t nread;
  nleft = count;
  while (nleft > 0) // La lettura è ripetuta nel ciclo fino all'esaurimento del numero di byte richiesti
    {

      if ((nread = read (fd, buf, nleft)) < 0)
        {
          if (errno == EINTR) // In caso di errore si controlla se è dovuto a un'interruzione della chiamata di sistema dovuta ad un segnale
            continue; // Ripeto l'accesso
        else
          return (nread); // Altrimenti l'errore viene ritornato al programma chiamante
          }
        else if (nread == 0) // Fine del file o socket chiusa
          return 0; // Si esce dal ciclo

        nleft -= nread; // Setta la variabile left per la funzione read
        buf += nread; // Setta il puntatore
    }
  return(nread);
  }

pid_t Fork()
  {
    pid_t pid;
    if ((pid = fork()) < 0)
    {
      perror("Fork Error");
      exit(1);
    }
    return pid;
}
int Select(int nfds,fd_set * readfds, fd_set * writefds,fd_set * errorfds,struct timeval * timeout)
{
  int readset;
  while((readset=select(nfds+1,readfds,(fd_set *)0,(fd_set *)0,NULL))<0)
   {

     if(errno==EINTR)
```

```
        {
         continue;
        }
       else
        {
          perror("select");
          exit(1);
        }
    }
    return readset;
}

#endif
```

HEADER.H

Di seguito sono illustrati la struttura dati utilizzata per rappresentare la blockchain e le firme di tutte le funzioni utilizzate, con una breve descrizione del loro scopo.

```c
#ifndef HEADER_H
#define HEADER_H
#include <stdlib.h>
#include <sys/types.h> /* predefined types */
#include <unistd.h> /* include unix standard library */
#include <arpa/inet.h> /* IP addresses conversion utililites */
#include <sys/socket.h> /* socket library */
#include <stdio.h> /* include standard I/O library */
#include <string.h>
#include <errno.h>
#include <time.h>
#include <syslog.h>
#include <stdbool.h>
#include "Wrapper.h"

typedef struct trax  //struttura che descrive la transazione
{
    int id_trax; //numero progressivo
    int sum; //ammontare
    struct sockaddr_in mittente, destinatario; //ip e porta del mittente e del destinatario
}Trax;

typedef struct blocco
{
    int id_b; //numero progressivo
    int tempo; //tempo di attesa
    Trax t; //transazione
}Blocco;

typedef struct nodo
{
    Blocco block;
    struct nodo *next;
}Nodo;

//Funzioni per la gestione della lista

//Crea un nuovo nodo da inserire in testa
void InserimentoNodo(Nodo *x, Nodo **Head);

//Crea un nuovo nodo da un blocco ricevuto
Nodo* CreaNodo(Blocco b);

//Funzione utile al BlockServer in quanto alla prima connessione con il NodoN riceve la
lista al "contrario", questa
//funzione ci permette di renderla uguale a quella del NodoN
void Reverse(Nodo** head);

//Funzione booleana che ritorna true se due indirizzi ip e porta sono uguali oppure no
bool Check(struct sockaddr_in key, struct sockaddr_in current);

//Dato un indirizzo ip-porta conta il numero di blocchi in cui è presente lo stesso
int RicercaIp(struct sockaddr_in key, Nodo* head);

//Permette di inviare il numero di blocchi in cui è coinvolto l'indirizzo ip:porta trovato
al Client
void ScritturaTransazioniIp(struct sockaddr_in key, Nodo* head, int fd);
```

```c
//Permette di inviare le N ultime transazioni al Client
void ScritturaNtransazioni(int num, int fd, Nodo* head);

//Funzione che ritorna il valore complessivamente scambiato con le transazioni memorizzate
int ValMax(Nodo *head);

//Funzione che ritorna true se l'id transazione esiste nella blockchain
int RicercaId(int id, Nodo* head);

//Funzione per inviare i dettagli di una specifica transazione ad un Client
void ScritturaDettagli(int id, Nodo* head, int fd);

//menù a video per il Client
void ShowMenu();

//Funzione attua a creare il solo Nodo Genesi utile per il NodoN
Nodo* NodoGenesi();

//Funzione che crea un blocco da input di tastiera
Blocco CreaBlocco();

//Funzione CLientEcho
void ClientEcho(FILE * filein, int socket);


//Funzione che dato un blocco stampa a video le informazioni
void LetturaBlocco(Blocco b);

//Ritorna un blocco errore identificato semplicemente dall'id del blocco 0
Blocco BloccoError();
```

```c
//Controlla che in un array di caratteri ci siano solo numeri
bool IsInt(char buffer []);
#endif
```

FUNCTION.C

```c
#include "header.h"
#define MAXLINE 1024

void InserimentoNodo(Nodo *x, Nodo **head)
{
    //Il nuovo nodo punterà alla testa corrente
    x->next=*head;
    //Il nuovo nodo diventa la testa
    *head=x;
}

Nodo* CreaNodo(Blocco b)
{
    //Alloco la memoria per un nuovo nodo
    Nodo* newNodo = (Nodo*)malloc(sizeof(Nodo));
    //Assegno il blocco al nuovo nodo
    newNodo->block=b;
    return newNodo;
}

void Reverse(Nodo** head)
{
    //Function in place per invertire la lista in place.
    Nodo* prev = NULL;
    Nodo* current = *head;
    Nodo* next= NULL;

    while(current!=NULL)
    {
        //Salvo il puntatore al prossimo elemento in una variabile
        next = current->next;
        //Imposto il puntatore al prossimo elemento all'elemento precedente (inizialmente
NULL)
        current->next = prev;
        //Salvo l'elemento corrente in una variabile per usarlo nella prossima iterazione
        prev=current;
        //Vado avanti nella lista
        current=next;
    }
    *head = prev;

}

int ValMax(Nodo *head)
{
    int somma=0;
    Nodo* temp=head;
    //Scrollo la lista e somme di tutte gli ammontare delle transazioni effettuate
    while(temp!=NULL)
    {
        somma += temp->block.t.sum;
        temp=temp->next;
    }

    return somma;
}


bool Check(struct sockaddr_in key, struct sockaddr_in current)
{
    //Controllo se l'indirizzo e la porta combaciano
```

```
    if (key.sin_addr.s_addr == current.sin_addr.s_addr && key.sin_port==current.sin_port)
        return true;
    else
        return false;
}

int RicercaIp(struct sockaddr_in key, Nodo* head)
{
    int num=0;
    Nodo* temp=head;
    while(temp!=NULL)
    {
        //Scrollo la lista e confronto la struttura sockaddr_in ricevuta in input sia col
destinatario che col mittente.
        if(Check(key,temp->block.t.mittente) == true || Check(key,temp-
>block.t.destinatario) == true)
            num++;
        temp=temp->next;
    }
    return num;
}


int RicercaId(int id, Nodo* head)
{
    Nodo* temp=head;
    //Scrollo la lista e cerco l'ID richiesto. Ritorno 1 se esiste.
    while(temp!=NULL)
    {
        if(temp->block.t.id_trax!=id)
            temp=temp->next;
        else
            return 1;
    }
    return 0;
}


void ScritturaTransazioniIp(struct sockaddr_in key, Nodo* head, int fd)
{
    int num=RicercaIp(key, head);
    Blocco x;
    Nodo* temp;
    //Invio al descrittore preso in input il numero di transazioni in cui è presente l'ip/
porta richiesto
    FullWrite(fd,&num,sizeof(int));
    if(num>0)
    {
        temp=head;
        while(temp!=NULL && num>0)
        {
            //Scrollo la lista, se IP e Porta del mittente o del destinatario sono uguali a
quelli presi in input li invio
            if(Check(key,temp->block.t.mittente) || Check(key,temp->block.t.destinatario))
            {
                x=temp->block;
                FullWrite(fd, &x, sizeof(Blocco));
                num--;
                temp=temp->next;
            }
            else //Il Nodo considerato non ha IP/PORTA del mittente/destinatario uguale a
quello in input.
            {
                temp=temp->next;
            }
```

```
        }
    }
}


void ScritturaNtransazioni(int num, int fd, Nodo* head)
{
    Nodo* temp=head;
    Blocco x;
    while(temp!=NULL && num>0) //finché ci sono blocchi da inviare
    {
        x=temp->block;
        //Invio i blocchi
        FullWrite(fd, &x, sizeof(Blocco));
        num--;
        temp=temp->next;
    }
}

void ScritturaDettagli(int id, Nodo* head, int fd)
{
    //Cerco se esiste una transazione con l'ID richiesto
    int esito=RicercaId(id, head);
    Nodo* temp=head;
    Blocco x;
    //Invio al Client l'esito della ricerca
    FullWrite(fd, &esito, sizeof(esito));
    //Se non è 0, allora esiste quella transazione e procedo ad inviare i dettagli
    if(esito!=0)
    {
        //Scrollo la lista cercando la transazione con l'id richiesto
        while(temp!=NULL)
            {
                if(temp->block.t.id_trax!=id)
                    temp=temp->next;
                else
                {
                    x=temp->block;
                    FullWrite(fd, &x, sizeof(Blocco));
                    break;          //L'id è univoco quindi dopo averlo trovato possiamo
interrompere il loop
                }
            }
    }
}

void ShowMenu()
{
    printf("\n1)Visualizzare le ultime n transazioni\n");
    printf("2)Visualizzare il valore complessivamente scambiato con le transazioni\n");
    printf("3)Visualizzare i dettagli di una transazione da richiere\n");
    printf("4)Visualizzare le transazioni in cui è coinvolto l'indirizzo ip da
richiere\n");
    printf("Inserire qualsiasi altro carattere per chiudere la connessione\n");
}


Nodo* NodoGenesi()
{
    //Creo un nodo fittizio con cui iniziare la blockchain
    Nodo *head=(Nodo*)malloc(sizeof(Nodo));
    head->block.t.id_trax=0;
    head->block.t.sum=0;
    //Unico parametro impostato ad 1, utile per poterlo identificare rispetto ad un blocco
Error
```

```
    head->block.id_b=1;
    head->block.tempo=0;
    head->block.t.mittente.sin_port=htons(0);
    head->block.t.destinatario.sin_port=htons(0);
    head->block.t.mittente.sin_family=AF_INET;
    head->block.t.destinatario.sin_family=AF_INET;
    inet_aton("0.0.0.1", (struct in_addr*)&head->block.t.mittente.sin_addr.s_addr);
    inet_aton("0.0.0.1",(struct in_addr*)&head->block.t.destinatario.sin_addr.s_addr);
    head->next=NULL;
    return head;
}

Blocco BloccoError()
{
    //Blocco creato nel caso ci sia un errore nell'inserimento dei dati da parte di
CreaBlocchi
    Blocco error;
    error.id_b=0;
    return error;
}


Blocco CreaBlocco()
{
    char buffer[MAXLINE];
    Blocco b;
    b.id_b=1;
    b.t.id_trax=1;
    /*Questi id sono temporanei, verranno corretti quando saranno mandati a NodoN */
    //Prendo in input tutti i parametri necessari per registrare una nuova transazione e
controllo siano validi
    printf("\nInserire ammontare:\n");
    fgets(buffer, MAXLINE, stdin);
    if(IsInt(buffer)!= true) //Controllo se è effettivamente un numero
    {
        printf("Impossibile inserire questo ammontare.\n");
        b=BloccoError();
        return b;
    }
    b.t.sum=atoi(buffer);
    printf("\nHai inserito: %d\n",atoi(buffer));
    printf("\nInserire ip mittente:\n");
    fgets(buffer,MAXLINE,stdin);
    if(inet_aton(buffer,(struct in_addr*)&b.t.mittente.sin_addr.s_addr)==0)
    {
        printf("Impossibile inserire questo indirizzo ip.\n");
        b=BloccoError();
        return b;
    }
    printf("\nHai inserito ip mittente: %s\n",buffer);
    printf("\nInserire porta mittente:\n");
    fgets(buffer,MAXLINE,stdin);
    if(IsInt(buffer)!= true)
    {
        printf("Impossibile inserire questa porta.\n");
        b=BloccoError();
        return b;
    }

    b.t.mittente.sin_port=htons(atoi(buffer));
    printf("\nHai inserito: %d\n",atoi(buffer));
    printf("\nInserire ip destinatario:\n");
    fgets(buffer,MAXLINE,stdin);
    if(inet_aton(buffer,(struct in_addr*)&b.t.destinatario.sin_addr.s_addr)==0)
    {
```

```
        printf("Impossibile inserire questo indirizzo ip.\n");
        b=BloccoError();
        return b;
    }
    printf("\nHai inserito ip destinatario: %s\n",buffer);
    printf("\nInserire porta destinatario:\n");
    fgets(buffer,MAXLINE,stdin);
    if(!IsInt(buffer))
    {
        printf("Impossibile inserire questa porta.\n");
        b=BloccoError();
        return b;
    }
    b.t.destinatario.sin_port=htons(atoi(buffer));
    return b;
}



void LetturaBlocco(Blocco b)
{
    printf("Id blocco: %d\n",b.id_b);
    printf("\nNumero transazione: %d\n", b.t.id_trax);
    printf("Ammontare: %d\n", b.t.sum);
    printf("Ip e porta mittente %s | %d\n",
inet_ntoa(b.t.mittente.sin_addr),ntohs(b.t.mittente.sin_port));
    printf("Ip e porta destinatario %s | %d\n",
inet_ntoa(b.t.destinatario.sin_addr),ntohs(b.t.destinatario.sin_port));
}

void ClientEcho(FILE * filein, int socket)
{
    int choice=0, intbuff, recbuff, controllo;
    struct sockaddr_in buff_add;
    char buffer [MAXLINE];
    Blocco temp;
    while(1)
    {
        ShowMenu();
        //Prendo in input la scelta del client
        fgets(buffer,MAXLINE,filein);
        choice=atoi(buffer);
        printf("Hai scelto: %d\n", choice);
        //Invio la richiesta del client al Server
        FullWrite(socket,&choice,sizeof(choice));
        switch(choice)
        {
            case 1: //Leggere un numero da tastiera, inviare la richiesta al BlockServer e
stampare a video i dati
                fputs("\nScrivere il numero di transazioni da leggere: \n", stdout);
                fgets(buffer,MAXLINE,filein);
                intbuff=atoi(buffer);
                FullWrite(socket,&intbuff,sizeof(intbuff));
                //È possibile che il numero di transazioni richiesto eccede quelli
effettivamente disponibili.
                //riceveremo un valore -1
                if (FullRead(socket,&controllo, sizeof(controllo))==0)
                {
                    printf("Connessione chiusa\n");
                    exit(-1);
                }
                if(controllo<0)
                    printf("Non ci sono %d transazioni da visualizzare.\n",intbuff);
                else
                {
```

```
                while(intbuff>0) //Se la mia richiesta è soddisfatta so quanti Blocchi
devo leggere
                {
                    //Leggo i blocchi richiesti
                    FullRead(socket,&temp,sizeof(Blocco));
                    //Stampo le transazioni
                    LetturaBlocco(temp);
                    intbuff--;
                }
            }
            break;
        case 2: //Visualizzare il valore complessivamente scambiato con le transazioni
            //Attendo il valore richiesto
            if (FullRead(socket,&recbuff,sizeof(recbuff)) == 0)
            {
                printf("Connessione chiusa\n");
                exit(-1);
            }
            printf("Valore complessivamente scambiato: %d\n",recbuff);
            break;
        case 3: //Visualizzare i dettagli di una transazione da richiere
            printf("Scrivere il numero della transazione di cui si vuole leggere le
informazioni: \n");
            //Prendo in input l'ID della transazione richiesta
            fgets(buffer, MAXLINE, filein);
            //Converto la stringa in un intero
            intbuff=atoi(buffer);
            //Invio l'id transazione
            FullWrite(socket,&intbuff,sizeof(intbuff));
            //Potremmo ricevere -1 se non esiste la transazione richiesta
            if (FullRead(socket,&recbuff, sizeof(recbuff))==0)
                                {
                printf("Connessione chiusa\n");
                exit(-1);
            }
            if(recbuff==0)
                printf("L'id richiesto non esiste.\n");
            else
                {
                    //Ricevo la transazione
                    FullRead(socket,&temp,sizeof(Blocco));
                    //La mostro a schermo
                    LetturaBlocco(temp);
                }
            break;
        case 4: //Visualizzare le transazioni in cui è coinvolto l'indirizzo ip da
richiere
            printf("Scrivere l'indirizzo ip di cui si vogliono conoscere le transazioni:
\n");
            //Prendo l'indirizzo IP
            fgets(buffer, MAXLINE, filein);
            //Lo metto in una struttura in_addr temporanea
            inet_aton(buffer, (struct in_addr*)&buff_add.sin_addr.s_addr);
            fputs("Scrivere la porta associata all'indirizzo ip di cui si vogliono
conoscere le transazioni: \n", stdout);
            fgets(buffer, MAXLINE, filein);
            //Inserisco la porta ricevuta in Input nella stessa struct temporanea
            buff_add.sin_port=htons(atoi(buffer));
            buff_add.sin_family=AF_INET;
            //Invio l'indirizzo al BlockServer
            FullWrite(socket, &buff_add,sizeof(buff_add));
            //Ricevo il quantitativo di blocchi che dovrò leggere
            if (FullRead(socket,&recbuff,sizeof(recbuff))==0)
            {
                printf("Connessione chiusa\n");
```

```
                    exit(-1);
                }
                if(recbuff==0) //Se non ci sono blocchi con quell'IP
                {
                    printf("Non ci sono transazioni in cui è coinvolto l'indirizzo
richiesto.\n");
                }
                else
                {
                    while(recbuff>0)
                    {   //Leggo i blocchi richiesti
                        FullRead(socket,&temp,sizeof(Blocco));
                        //Li mostro a video
                        LetturaBlocco(temp);
                        recbuff--;
                    }
                }
                break;
            default:
                //Se è stato ricevuto un input <1 o >4 la connessione si chiude
                printf("\nConnessione chiusa.\n");
                close(socket);
                exit(1);
                break;
        }
    }
}

bool IsInt(char buffer [])
{
    int i;
    for(i=0; i<strlen(buffer)-1; i++)
    {
        //Controllo carattere per carattere se è un numero da 0 a 9
        if(buffer[i]<48 || buffer[i]>57)
            return false;
    }
    return true;
}
```

NODON.C

```c
#include "function.c"

int main(int argc, char *argv[])
{
    srand(time(NULL)); //Creazione seed per il tempo di attesa random

    //Creazione nodo Genesi
    Nodo *head = NodoGenesi();
    int list_fd, conn_fd=-1, creablocchi_fd, i, buffnum, numBlocco,enable=1;
    struct sockaddr_in serv_addr, c_addr, cb_addr;
    socklen_t len;            //dimensione di una socket
    ssize_t nread;
    fd_set f_set;
    int max_fd;      //numero massimo di descrittori che la select deve monitorare
    int segnale; //variabile
    char buffer[MAXLINE];
    Blocco temp;

    //Creazione socket per la listen
    list_fd=Socket(AF_INET,SOCK_STREAM,0);
    serv_addr.sin_family=AF_INET;
    //Prendere da linea di comando porta e ip
    serv_addr.sin_port=htons(1025);
    inet_aton("127.0.0.1", (struct in_addr*)&serv_addr.sin_addr.s_addr);

    setsockopt(list_fd,SOL_SOCKET,SO_REUSEADDR,&enable,sizeof(enable)); //permette il
riutilizzo di indirizzi locali

    //Assegniamo la listen all'indirizzo
    Bind(list_fd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    //Mettiamo in ascolto il NodoN
    Listen(list_fd, 1025);
    puts("Attendo la connessione del creatore di blocchi.\n");
    creablocchi_fd = Accept(list_fd, (struct sockaddr*)&cb_addr, &len);
    //Se la Accept è riuscita
    if (creablocchi_fd != -1)
    puts("Connesso!\n");
    //Rispondo al CreaBlocchi che ci siamo connessi
    FullWrite(creablocchi_fd,&segnale,sizeof(segnale));



    while(1)
    {

        FD_ZERO(&f_set);                //pulisce f_set
        FD_SET(creablocchi_fd,&f_set);  //CreaBlocchi.c
        FD_SET(list_fd,&f_set);
        if (list_fd > creablocchi_fd)
            max_fd=list_fd;
        else
            max_fd = creablocchi_fd;
        if(conn_fd>0)                   //se il BlockServer si è connesso
        {
            FD_SET(conn_fd,&f_set); //lo aggiungo alla select nel nuovo ciclo
            if(max_fd<conn_fd)//aggiorno max_fd
                max_fd=conn_fd;

        }


        Select(max_fd+1,&f_set,NULL,NULL,NULL);
```

```
        if(FD_ISSET(creablocchi_fd,&f_set))    //Se ci sono nuovi blocchi da aggiungere
alla blockchain
        {
            //ricevo il segnale
            if(FullRead(creablocchi_fd,&segnale, sizeof(segnale))==0)
            {
                printf("Connessione con Crea Blocchi caduta, chiudo NodoN\n");
                exit(-1);
            }
            //Invio la conferma della ricezione del segnale
            FullWrite(creablocchi_fd,&segnale, sizeof(segnale));
            //Aspetto il blocco
            FullRead(creablocchi_fd,&temp,sizeof(temp));
            temp.id_b = 1 + head->block.id_b;  //Assegno al blocco ricevuto un
Identificativo
            temp.t.id_trax = 1 + head->block.t.id_trax; //Assegno alla transazione un
identificativo
            temp.tempo=(rand()%10)+5; //Assegno un tempo di attesa casuale
            printf("Tempo di attesa: %d\n",temp.tempo);
            sleep(temp.tempo);
            InserimentoNodo(CreaNodo(temp),&head);
            printf("Ho ricevuto un nuovo blocco\n");
            LetturaBlocco(head->block);   //Visualizzo il nuovo blocco ricevuto
            if (conn_fd > 0) //Se il BlockServer è attivo
                {FullWrite(conn_fd,&temp,sizeof(temp));} //invio il nuovo blocco al
BlockServer

        }


        if (FD_ISSET(list_fd,&f_set))
        {
            len=sizeof(c_addr);
            conn_fd=Accept(list_fd, (struct sockaddr*)&c_addr, &len);
            //Inserisco il BlockServer tra i descrittori da controllare nella Select
            FD_SET(conn_fd, &f_set);
            if(max_fd<conn_fd)
                max_fd=conn_fd;
            //Mi assicuro che BlockServer sia effettivamente connesso
            FullWrite(conn_fd,&segnale, sizeof(segnale));
            FullWrite(STDOUT_FILENO,"\nE' stata effettuata una nuova connessione con il
Blockserver\n",sizeof("\nE' stata effettuata una nuova connessione con il Blockserver\n"));

        }

        //Controllo se il descrittore del BlockServer è attivo solo se è connesso.
        if(conn_fd>0)
        {
        if(FD_ISSET(conn_fd,&f_set))
        {
                if (FullRead(conn_fd,&numBlocco,sizeof(numBlocco)) == 0) //Se il BlockServer
è caduto
                    {
                        if (max_fd == conn_fd) //Aggiorno il max_fd
                        {
                            if (list_fd > creablocchi_fd)
                                max_fd=list_fd;
                            else
                                max_fd=creablocchi_fd;
                        }
                        printf("Il Blockserver si è disconnesso.\n");
                        //Chiudo la connessione e reimposto il descrittore a -1 in modo da
poter accettare una nuova connessione
```

```
                        close(conn_fd);
                        conn_fd=-1;

                    }
                    else
                    {
                    if (Fork()==0) //Sono il figlio
                    {
                        close(creablocchi_fd);
                        close(list_fd);

                        printf("Ultimo blocco del Blockserver connesso: %d\n",numBlocco);
                        buffnum=head->block.id_b - numBlocco;
                        if (numBlocco > buffnum)
                        printf("Non è stato inviato nessun nuovo nodo\n");

                        FullWrite(conn_fd,&buffnum,sizeof(buffnum));
                        //Mi accerto che ha letto quanti ne sto per inviare

                        if(buffnum>0)
                        {
                            ScritturaNtransazioni(buffnum,conn_fd,head); //invio tutte le
transazioni richieste dal BlockServer
                        }
                    close(conn_fd);
                    return 0;
                    }
            }
        }
        }


    }
    return 0;
}
```

CREABLOCCHI.C

```c
#include "function.c"

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

    int fd, conn, enable=1;
    char buffer [MAXLINE];
    struct sockaddr_in nodon_add;
    Blocco temp;
    int tempo, nleft;
    int segnale=1;
    //Creazione socket per il collegamento con Nodon
    fd=Socket(AF_INET, SOCK_STREAM,0);
    //Imposto l'indirizzo
    nodon_add.sin_family=AF_INET;
    inet_aton(argv[1],(struct in_addr*)&nodon_add.sin_addr.s_addr);
    nodon_add.sin_port=htons(atoi(argv[2]));
    //Imposto che l'indirizzo è riutilizzabile
    setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&enable,sizeof(enable));
    //Mi connetto al NodoN
    Connect(fd,(struct sockaddr*)&nodon_add,sizeof(nodon_add));
    //Aspetta una risposta dal NodoN
    FullRead(fd,&segnale,sizeof(segnale));

    while(1)
    {
        //Creiamo il blocco e lo mettiamo in testa
            temp=CreaBlocco();
            if(temp.id_b!=0) //Non è un blocco error
            {
                FullWrite(fd,&segnale,sizeof(segnale)); //Comunico al NodoN che sto per
inviargli un blocco
                //aspetto la conferma della ricezione del segnale
                if (FullRead(fd,&segnale,sizeof(segnale)) == 0)
                    {
                        printf("NodoN è Disconnesso, chiusura del CreaBlocchi.\n");
                        exit(-1);
                    }
                //Invio il nuovo blocco
                FullWrite(fd,&temp,sizeof(temp));
                printf("Inviato al NodoN\n");
            }
            else
            {
                printf("Errore inserimento blocco\n");;
            }
    }
    return 0;
}
```

BLOCKSERVER.C

```c
#include "function.c"

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

    int list_fd, conn_fd, nodon_fd, blocchiInit, controllo;
    int i, nread, enable=1, max_fd, choice = 1, recbuff, sendbuff,segnale=1;
    struct sockaddr_in serv_add, client, nodon_add, buff_add;
    socklen_t len;
    char buffer[MAXLINE];
    fd_set f_set;
    pid_t pid;
    Nodo *head;
    Blocco temp, temp2;
    //Creo un descrittore per la Listen
    list_fd = Socket(AF_INET,SOCK_STREAM,0);
    //Impostazioni dell'indirizzo del BlockServer
    //Imposto la famiglia dell'indirizzo
    serv_add.sin_family=AF_INET;
    //Imposto la porta
    serv_add.sin_port=htons(1026);
    //Imposto l'indirizzo IP
    inet_aton("127.0.0.1", (struct in_addr*)&serv_add.sin_addr.s_addr);
    //Setto l'opzione per il riutilizzo dello stesso indirizzo
    setsockopt(list_fd,SOL_SOCKET,SO_REUSEADDR,&enable,sizeof(enable));
    //Associo l'indirizzo con la socket
    Bind(list_fd,(struct sockaddr*)&serv_add,sizeof(serv_add));
    //definisco la lunghezza della coda di attesa
    Listen(list_fd,1026);
    //Creo un descrittore per il NodoN a cui connettermi
    nodon_fd = Socket(AF_INET,SOCK_STREAM,0);
    //Imposto la famiglia
    nodon_add.sin_family=AF_INET;
    //Imposto la porta presa in input
    nodon_add.sin_port=htons(atoi(argv[2]));
    //Imposto l'indirizzo preso in input
    if(inet_pton(AF_INET,argv[1],(struct sockaddr*)&nodon_add.sin_addr)  <0)
    {
        fprintf(stderr,"Errore inet_pton %s\n", argv[1]);
        exit(1);
    }
    //Prendo in input l'identificativo dell'ultimo blocco registrato dal Blockserver
    blocchiInit = atoi(argv[3]);
    if (blocchiInit <0)
        {
            printf("Id blocco non valido\n");
            exit(-1);
        }
    //Connessione al NodoN
    Connect(nodon_fd,(struct sockaddr*)&nodon_add, sizeof(nodon_add));
    //Dico al NodoN che sono connesso
    FullRead(nodon_fd,&segnale,sizeof(int));
    //Invio il numero di blocchi iniziali
    FullWrite(nodon_fd, &blocchiInit, sizeof(blocchiInit));
    //Aspetto il numero di blocchi da dover leggere
    FullRead(nodon_fd, &nread, sizeof(nread));
    //Ho ricevuto il numero di blocchi che dovrò leggere
    for(i=0; i<nread; i++)
    {
        //Leggo un blocco
        FullRead(nodon_fd, &temp, sizeof(temp));
         //Lo inserisco in un nodo e poi lo inserisco in testa alla lista
        if(i==0)
```

```
    {
        //Se è la prima iterazione fa l'assegnamento della testa.
        head=CreaNodo(temp);
    }
    else
        //Inserisce in testa
        InserimentoNodo(CreaNodo(temp),&head);
}
if (nread >0)
printf("Mi sono connesso ed ho ricevuto %d blocchi\n",nread);
else
    printf("Non ho ricevuto nuovi blocchi\n");
//Ho ricevuto la Blockchain ma inversa (1-2-3) quindi la capovolgo (3-2-1)
Reverse(&head);


while(1)
{
    //Impostazioni del set di descrittori da controllare
    FD_ZERO(&f_set);
    FD_SET(nodon_fd,&f_set);
    FD_SET(list_fd, &f_set);
    //
    if (nodon_fd > list_fd)
        max_fd=nodon_fd;
    else
        max_fd = list_fd;

    Select(max_fd,&f_set,NULL,NULL,NULL);

    if(FD_ISSET(nodon_fd,&f_set))
    {
        //Se ho ricevuto una nuova connessione con nodoN
        if (FullRead(nodon_fd,&temp,sizeof(temp)) == 0) //Se ho ricevuto un segnale
di chiusura
        {
            //Esco, il BlockServer non può vivere se il NodoN è chiuso
            printf("Persa la connessione con NodoN, chiudere il BlockServer.\n");
            close(nodon_fd);
            exit (-1);
        }
        //Lo inserisco in testa alla blockchain.
        InserimentoNodo(CreaNodo(temp),&head);
        printf("Inserito un nuovo blocco con ID: %d\n",head->block.id_b);
    }

    if(FD_ISSET(list_fd,&f_set))
    {
    //Nuova connessione con un Client
    len=sizeof(client);
    conn_fd=Accept(list_fd,(struct sockaddr*)&client,&len);
    if(Fork()==0) //Sono il figlio
    {
        close(list_fd);
        close(nodon_fd);


        //Dico al Client che sono connesso
        FullWrite(conn_fd,&segnale,sizeof(segnale));
        inet_ntop(AF_INET,&client.sin_addr,buffer,sizeof(buffer));
        printf("Nuovo client connesso: IP %s, port %d\n",buffer,ntohs(client.sin_port));
        //Choice è inizializzata ad 1, quindi entrerà nel while la prima volta
        while (choice > 0 && choice < 5)
        {
            //Considero la scelta del client
```

```
                if (FullRead(conn_fd,&choice,sizeof(choice)) == 0) //Se la connessione è
caduta
                {
                    printf("Connessione chiusa: IP %s, port
%d\n",buffer,ntohs(client.sin_port));
                    close(conn_fd);
                    return 0;
                }
                switch(choice)
                {
                    case 1:
                        FullRead(conn_fd,&recbuff,sizeof(recbuff)); //Il numero di
transazioni da voler leggere
                        // controllo se posso inviarli
                        controllo=head->block.id_b - recbuff; //differenza fra la richiesta
e quello che ho
                        printf("Numero transazioni da scrivere a IP %s, porta %d valore:
%d\n",buffer, ntohs(client.sin_port),recbuff);
                        FullWrite(conn_fd,&controllo,sizeof(controllo)); //Invio il
controllo che mi dice se posso soddisfare la richiesta

                        if(controllo>=0)
                        {                       //Se la differenza è maggiore o uguale di
zero significa che ho sufficienti numero di blocchi
                            ScritturaNtransazioni(recbuff,conn_fd,head);
                        }
                        break;
                    case 2:
                        //Richiesta dell'ammontare totale scambiato all'interno della
blockchain
                        //Calcolo la somma totale
                        sendbuff=ValMax(head);
                        printf("Invio a IP %s, porta %d valore:
%d\n",buffer,ntohs(client.sin_port),sendbuff);
                        //Invio la somma totale
                        FullWrite(conn_fd,&sendbuff,sizeof(sendbuff));
                        break;
                    case 3:
                        //Ricevo l'id della transazione da visualizzare
                        FullRead(conn_fd,&recbuff, sizeof(recbuff));
                        //Invio la transazione richiesta
                        ScritturaDettagli(recbuff, head, conn_fd);
                        break;
                    case 4:
                        //Ricevo l'indirizzo di cui vuole avere informazioni
                        FullRead(conn_fd,&buff_add,sizeof(buff_add));
                        //La function invierà prima il numero delle transazioni che
rispettano la richiesta e poi le suddette transazioni
                        ScritturaTransazioniIp(buff_add,head,conn_fd);
                        break;
                } //fine switch
            }//fine while
                //Finite le richiesta
                printf("Connessione chiusa: IP %s, porta %d\n",buffer,ntohs(client.sin_port));
                close(conn_fd);
                return 0;

            }
            else
            {
                /* padre */
                close(conn_fd);
            }
        }
        }
```

```
    return 0;
}
```

CLIENT.C

```c
#include "function.c"


int main(int argc, char *argv[])
{
    int sock, enable=1, segnale=1;
    struct sockaddr_in serv_add;
    //Creo il descrittore per la connessione con il BlockServer
    sock = Socket(AF_INET,SOCK_STREAM,0);
    //Imposto la famiglia
    serv_add.sin_family=AF_INET;
    //Imposto la porta da input
    serv_add.sin_port=htons(atoi(argv[2]));
    //Imposto l'IP da input
    if(inet_pton(AF_INET,argv[1],(struct sockaddr*)&serv_add.sin_addr)  <0)
    {
        fprintf(stderr,"Errore inet_pton %s\n", argv[1]);
        exit(1);
    }
    //Imposto la riusabilità dell'indirizzo associato
    setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&enable,sizeof(enable));
    //Mi connetto al BlockServer
    Connect(sock,(struct sockaddr*)&serv_add, sizeof(serv_add));
    //Ricevo un segnale
    FullRead(sock,&segnale, sizeof(segnale));
    //Avvio la funzione che gestisce la connessione con il BlockServer
    ClientEcho(stdin,sock);
}
```

MANUALE UTENTE

Indichiamo qui di seguito le istruzioni per la compilazione e l'esecuzione di tutti i file descritti, specificando anche l'ordine di esecuzione.

COMPILAZIONE

- **CreaBlocchi**: gcc CreaBlocchi.c -o cb
- **NodoN**: gcc NodoN.c -o nodon
- **BlockServer**: gcc BlockServer.c -o bs
- **Client**: gcc Client.c -o cl

ESECUZIONE

1. **NodoN**: ./nodon
2. **CreaBlocchi**: ./cb 127.0.0.1 1025
3. **BlockServer**: ./bs 127.0.0.1 1025 (È da aggiungere il numero di blocchi iniziali)
4. **Client**: ./cl 127.0.0.1 1026

SIMULAZIONE

È disponibile una video simulazione di un'esecuzione tipo.