UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA E COMUNICAZIONE

Laboratorio Reti di Calcolatori

Laurea Triennale in Comunicazione Digitale

Docente: Matteo Zignani

Lezione 5

Anno accademico 2011/12

Implementazione server multithread

Processi e thread

- Sistemi operativi multitask: più entità software fanno uso concorrente delle risorse
- Task = Processo (programma associato ad un contesto)
- Linguaggi programmazione moderni gestiscono flussi esecutivi paralleli. Processi o thread in funzione del SO e del linguaggio di programmazione.
- Processo = ambiente di esecuzione a cui associo un insieme di risorse private (spazio di indirizzamento)
  - Comunicazione tra processi attraverso *Inter Process Communication* (pipe, socket)
  - In Java la creazione dei processi viene demandata ad un'istanza della classe *ProcessBuilder*

Thread

- Thread o processo leggero
  - Richiede meno risorse
  - Esistono all'interno di un processo. Condividono le risorse di un processo (memoria, file aperti)
  - Granularità più fine: processo può suddividere la sua esecuzione in thread

Definizione di un thread

- Due modi per definire thread in Java
  1. Creo un oggetto che implementa l'interfaccia *Runnable*. C'è solo un metodo da implementare: *run()*. Oggetto passato al costruttore della classe *Thread*.

```
1  public class HelloRunnable implements Runnable {
2                  EserciziLab/src/Server.java
3      public void run() {
4          System.out.println("Metodo run()");
5      }
6
7      public static void main(String args[]) {
8          (new Thread(new HelloRunnable())).start();
9      }
10
11 }
12
```

Definizione di un thread(2)

2. Creo un oggetto che estende la classe *Thread*. Classe *Thread* implementa *Runnable* ma metodo *run()* non fa niente.

```
1  public class HelloThread extends Thread {
2
3      public void run() {
4          System.out.println("Metodo run()!");
5      }
6
7      public static void main(String args[]) {
8          (new HelloThread()).start();
9      }
10
11 }
```

- POI in entrambi i casi invoco metodo *start()* della classe *Thread*

Thread in pausa e interrupt

- Thread.sleep() sospende l'esecuzione per un tempo specificato
  - *sleep(long millisec)*
  - *sleep(long millisec, int nanosec)*
    - Non c'è precisione dato che dipendondo da S.O.
- Periodo di sleep interrotto con interrupt
- *InterruptedException*: altro thread interrompe il thread corrente
- **Interrupt**: segnale ad un thread che dovrebbe fermarsi la sua esecuzione.
    - Programmatore decide la risposta ad interrupt. Di solito termino thread
  - Si chiama metodo *interrupt()* dell'oggetto a cui voglio inviare interrupt
  - Il thread a cui invio interrupt deve supportare il segnale
    - Se molti metodi che gettano *InterruptedException* prendo l'eccezione
    - Periodicamente invoco il metodo *interrupted()*
  - Interrupt status flag

Thread in pausa e interrupt(2)

```
ublic class SleepMessages {
    public static void main(String args[]) throws InterruptedException {
        String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
        };

        for (int i = 0; i < importantInfo.length; i++) {
            try{
            //Pausa per 4 secondi
            Thread.sleep(4000);}
            catch(InterruptedException ie){
                return;
            }
            //Scrivi il messaggio i-esimo dall'array
            System.out.println(importantInfo[i]);
        }
    }
}
```

```
public class SleepMessages {
    public static void main(String args[]) throws InterruptedException {
        String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
        };

        for (int i = 0; i < importantInfo.length; i++) {
            System.out.println(importantInfo[i]);
            if(Thread.interrupted()){
                throw new InterruptedException();
            }
        }
    }
}
```

- Interrupted() / isInterrupted()

Join

- t.join(): il thread corrente si mette in pausa fino a che il thread t finisce

```
blic class SimpleThreads {

    //Display a message, preceded by the name of the current thread
    static void threadMessage(String message) {
        String threadName = Thread.currentThread().getName();
        System.out.format("%s: %s%n", threadName, message);
    }

    private static class MessageLoop implements Runnable {
        public void run() {
            String importantInfo[] = {
                "Mares eat oats",
                "Does eat oats",
                "Little lambs eat ivy",
                "A kid will eat ivy too"
            };
            try {
                for (int i = 0; i < importantInfo.length; i++) {
                    //Pause for 4 seconds
                    Thread.sleep(4000);
                    //Print a message
                    threadMessage(importantInfo[i]);
                }
            } catch (InterruptedException e) {
                threadMessage("I wasn't done!");
            }
        }
    }
```

```
    public static void main(String args[]) throws InterruptedException {

        //Delay, in milliseconds before we interrupt MessageLoop
        //thread (default one hour).
        long patience = 1000 * 60 * 60;

        //If command line argument present, gives patience in seconds.
        if (args.length > 0) {
            try {
                patience = Long.parseLong(args[0]) * 1000;
            } catch (NumberFormatException e) {
                System.err.println("Argument must be an integer.");
                System.exit(1);
            }

        }

        threadMessage("Starting MessageLoop thread");
        long startTime = System.currentTimeMillis();
        Thread t = new Thread(new MessageLoop());
        t.start();

        threadMessage("Waiting for MessageLoop thread to finish");
        //loop until MessageLoop thread exits
        while (t.isAlive()) {
            threadMessage("Still waiting...");
            //Wait maximum of 1 second for MessageLoop thread to
            //finish.
            t.join(1000);
            if (((System.currentTimeMillis() - startTime) > patience) &&
                  t.isAlive()) {
                threadMessage("Tired of waiting!");
                t.interrupt();
                //Shouldn't be long now -- wait indefinitely
                t.join();
            }

        }
        threadMessage("Finally!");
    }
```

Inisidie nell'uso dei thread

1. Thread interfence: errori introdotti quando più thread accedono a dati condivisi e vi eseguono operazioni che consistono in più passi

```
1  class Counter {
2      private int c = 0;
3
4      public void increment() {
5          c++;
6      }
7
8      public void decrement() {
9          c--;
10     }
11
12     public int value() {
13         return c;
14     }
15
16 }
```

Thread A invoca increment()

Thread B invoca decrement()

2. Memory Consistency Error: thread diversi hanno viste incosistenti su un dato che dovrebbe essere uguale per entrambi. Problema risolvibile con relazione happens-before

Sincronizzazione

- Risolve i precedenti problemi.
- Costruita attorno ad entità interna: Monitor Lock associato ad ogni oggetto. Thread che vuole accesso esclusivo deve prendere il lock prima di accedervi e rilasciarlo quando finisce.
- Livello metodo: dichiaro il metodo *synchronized*

```
1  public class SynchronizedCounter {
2      private int c = 0;
3
4      public synchronized void increment() {
5          c++;
6      }
7
8      public synchronized void decrement() {
9          c--;
10     }
11
12     public synchronized int value() {
13         return c;
14     }
15 }
```

Altri thread si bloccano fino al termine del metodo

Sincronizzazione (2)

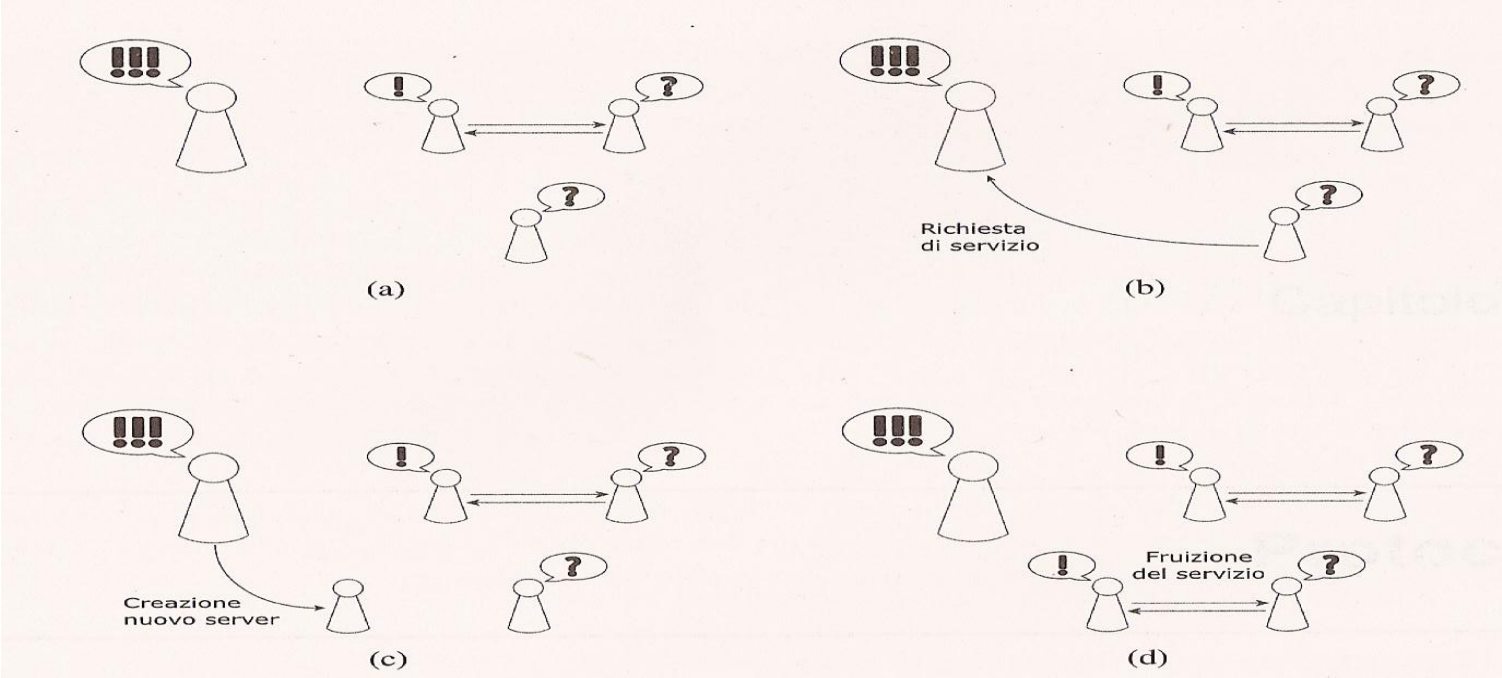
- Livello statement
  - Sincronizzazione con grana più fine
  - Devo specificare l'oggetto che fornisce il lock

```
public void addName(String name) {
    synchronized(this) {
        lastName = name;
        c++;
    }
    nameList.add(name);
}
```

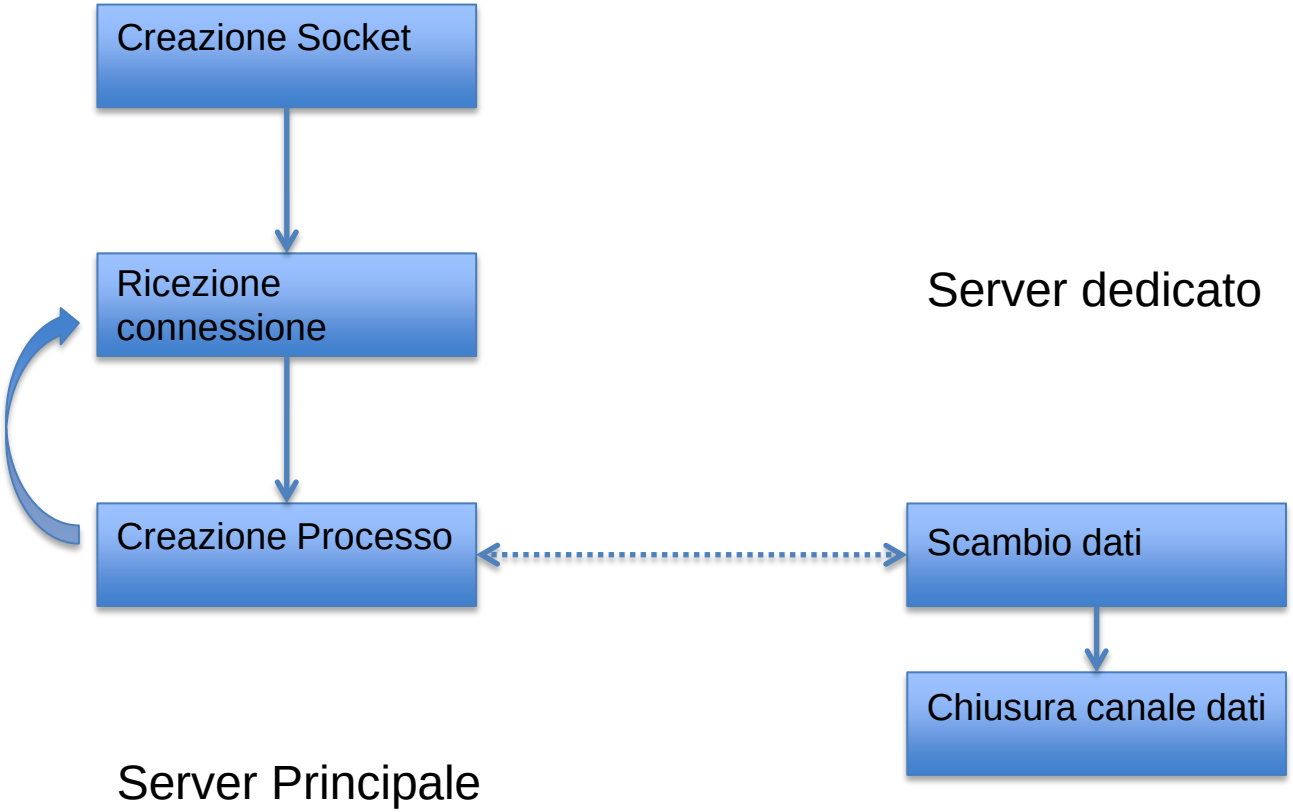
```
1  public class MsLunch {
2      private long c1 = 0;
3      private long c2 = 0;
4      private Object lock1 = new Object();
5      private Object lock2 = new Object();
6
7      public void inc1() {
8          synchronized(lock1) {
9              c1++;
10         }
11     }
12
13     public void inc2() {
14         synchronized(lock2) {
15             c2++;
16         }
17     }
18 }
```

Server Multithread

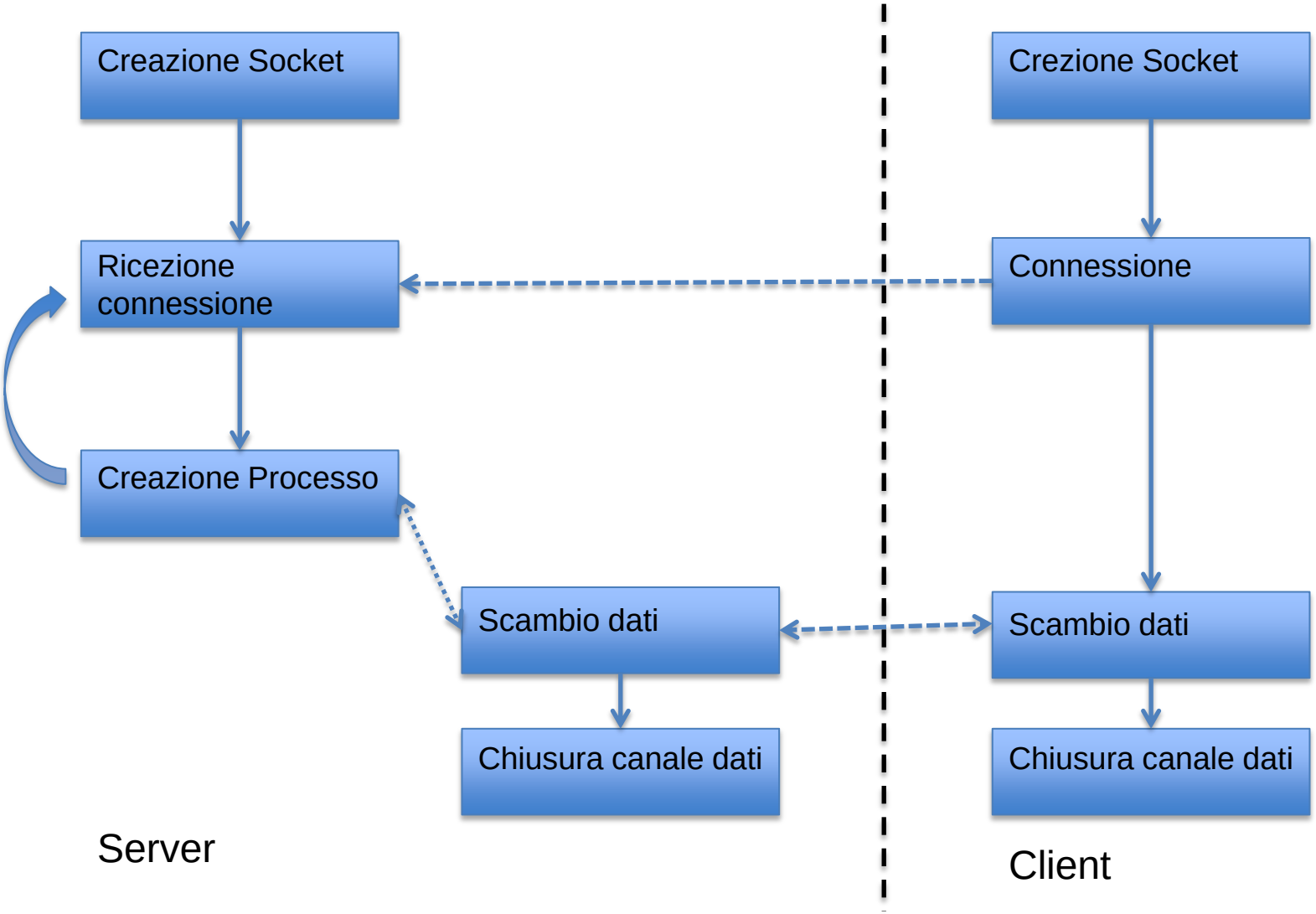
- Server principale gestisce le chiamate
  - Non gestisce direttamente lo scambio dati
- Delega un nuovo thread ad occuparsi della gestione del servizio



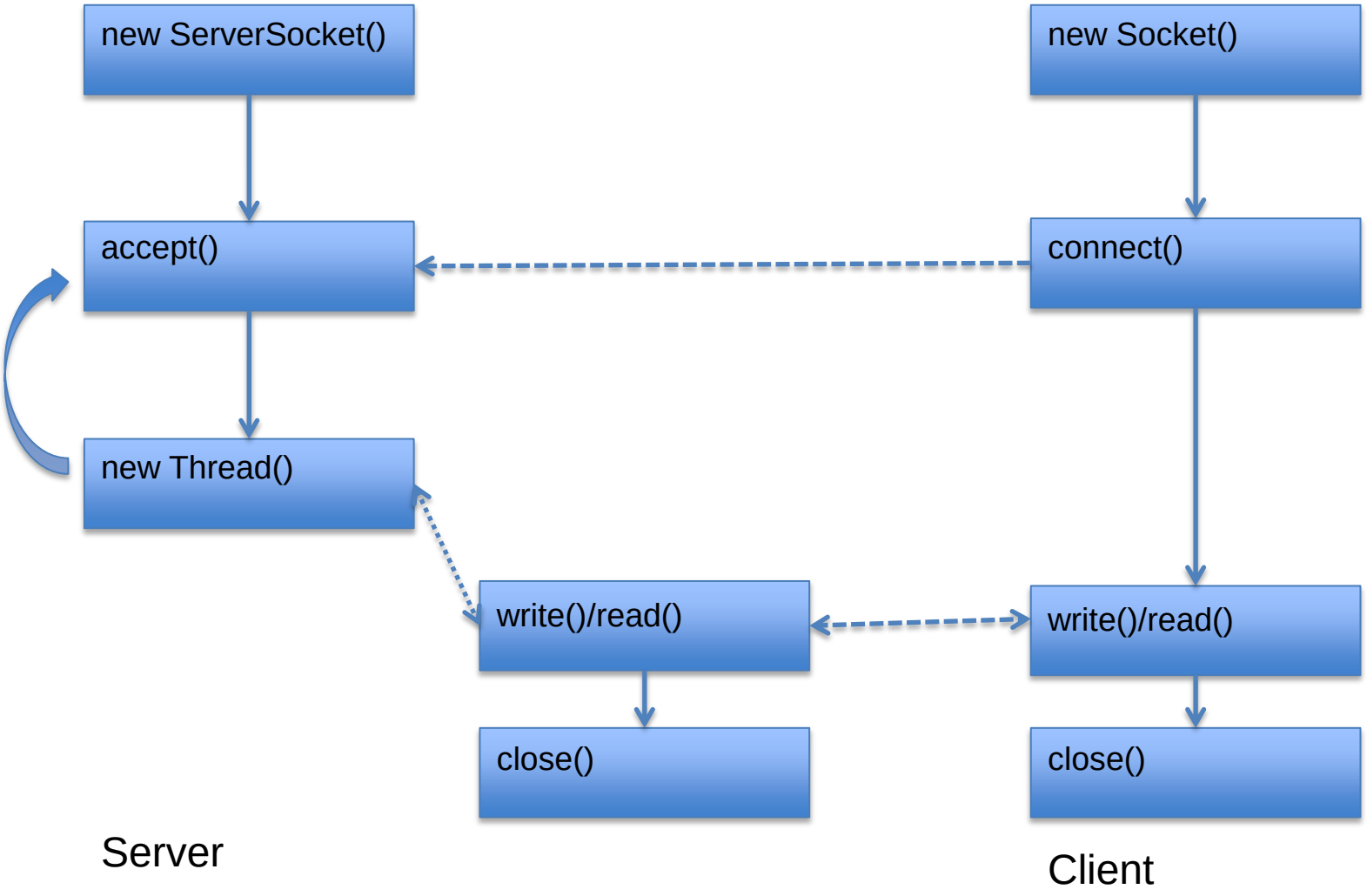
Schema del server



Operazioni server multithread/client



In Java


new ServerSocket()
new Socket()
accept()
connect()
new Thread()
write()/read()
write()/read()
close()
close()
Server
Client

Server con thread esterno

```
public class ServerMultiThread {

    public static void main(String[] args) {
        try{
            int port = 0;
            if (args.length > 1){
                throw new IllegalArgumentException("numero parametri non corretto");
            }
            if (args.length == 1){
                port = Integer.parseInt(args[0]);
                if (port < 0){
                    throw new IllegalArgumentException("porta non valida");
                }
            }
            ServerSocket s = new ServerSocket(port);
            if(port == 0){
                System.out.println("Porta allocata: " + s.getLocalPort());
            }
            while (true){
                Socket data_socket = s.accept();
                Thread t = new ErogazioneServizio(data_socket);
                t.start();
                s.close();
            }

        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

}
```

Thread per la gestione

```
public class ErogazioneServizio extends Thread{

    private Socket socket;

    public ErogazioneServizio(Socket s){
        socket = s;
    }

    public void run(){
        try{
            int dim_b = 1024;
            byte[] buffer = new byte[dim_b];
            int r;
            InputStream is = socket.getInputStream();
            OutputStream os = socket.getOutputStream();
            while(true){
                r = is.read(buffer, 0, dim_b);
                if (r>0){
                    os.write(buffer, 0, r);
                }
                else return;
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }

    }

}
```

Server in una sola classe

```
6  public class HelloServer extends Thread{
7
8      public static void main(String[] args) {
9          try{
10             ServerSocket ss = new ServerSocket(0);
11             System.out.println("Server sulla porta: " + ss.getLocalPort());
12             while(true){
13                 Socket s = ss.accept();
14                 (new HelloServer(s)).start();
15             }
16         }
17         catch(Exception e){
18             e.printStackTrace();
19         }
20     }
21
22     private Socket socket;
23     private String messaggio;
24
25     public HelloServer(Socket s){
26         socket = s;
27         messaggio = "Sono il thread: "+this.getId();
28     }
29
30     public void run(){
31         try{
32             OutputStream os = socket.getOutputStream();
33             os.write(messaggio.getBytes());
34             socket.close();
35         }
36         catch(Exception e){
37             e.printStackTrace();
38         }
39     }
40
41 }
```

Esempio di sincronizzazione

```java
import java.net.ServerSocket;
import java.net.Socket;


public class ServerSync {

    public static void main(String[] args){
        Counter c = new Counter();
        try{
            ServerSocket ss = new ServerSocket(0);
            System.out.println("Server port: " + ss.getLocalPort());
            while(true){
                Socket s = ss.accept();
                (new ThreadSync(s,c)).start();
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

```java
public class Counter {
    private int c;
     public Counter(){
         c=0;
     }

     public synchronized int getNextValue(){
         c++;
         return c;
     }
}

```

Esempio di sincronizzazione(2)

```
import java.io.OutputStream;
import java.net.Socket;


public class ThreadSync extends Thread {

    private Socket s;
    private String messaggio;

    public ThreadSync(Socket s, Counter c){
        this.s = s;
        messaggio = "il thread "+getId()+" è il "+c.getNextValue()+" thread invocato dal server";
    }

    public void run(){
        try{
            OutputStream os = s.getOutputStream();
            os.write(messaggio.getBytes());
            s.close();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }


}
```

Esercizi

1. Si implementi un client multithread per il servizio echo che attenda i dati in parallelo da
   - Socket con cui client collegato al server
   - Stream associato alla tastiera
2. Rendere il server chargen multithread
3. Modificare il server della slide 17 in modo tale che accetti client che inviano la giusta parola segreta. In caso positivo il servizio viene, come al solito, affidato ad un thread per lo scambio dati
4. Implementare un server multithread che, quando contattato, mandi al client una serie di stringhe contenenti la sequenza dei numeri naturali
   - Stringhe inviate ad intervalli regolari di un secondo
   - L'output non termina mai

Esercizi(2)

5. Modificare il server precedente in modo tale che ci sia un thread separato che ad intervalli regolari di 3 sec incrementi un contatore che viene letto dagli altri thread (clients) ad intervalli regolari di un sec
6. Si implementi un server multithread che gestisce una rubrica di nomi (quale struttura dati posso usare?). Il client può inviare due comandi
   - i:<nome> = inserisce un nome nella rubrica
   - d:<nome> = elimina un nome nella rubrica
   - q = chiude la connessione

   Il server deve gestire tutti i casi possibili e comunicare al client se l'operazione ha avuto esito positivo o ci sono stati problemi