Ingegneria del Software

Threads

Mauro Migliardi Ph. D.

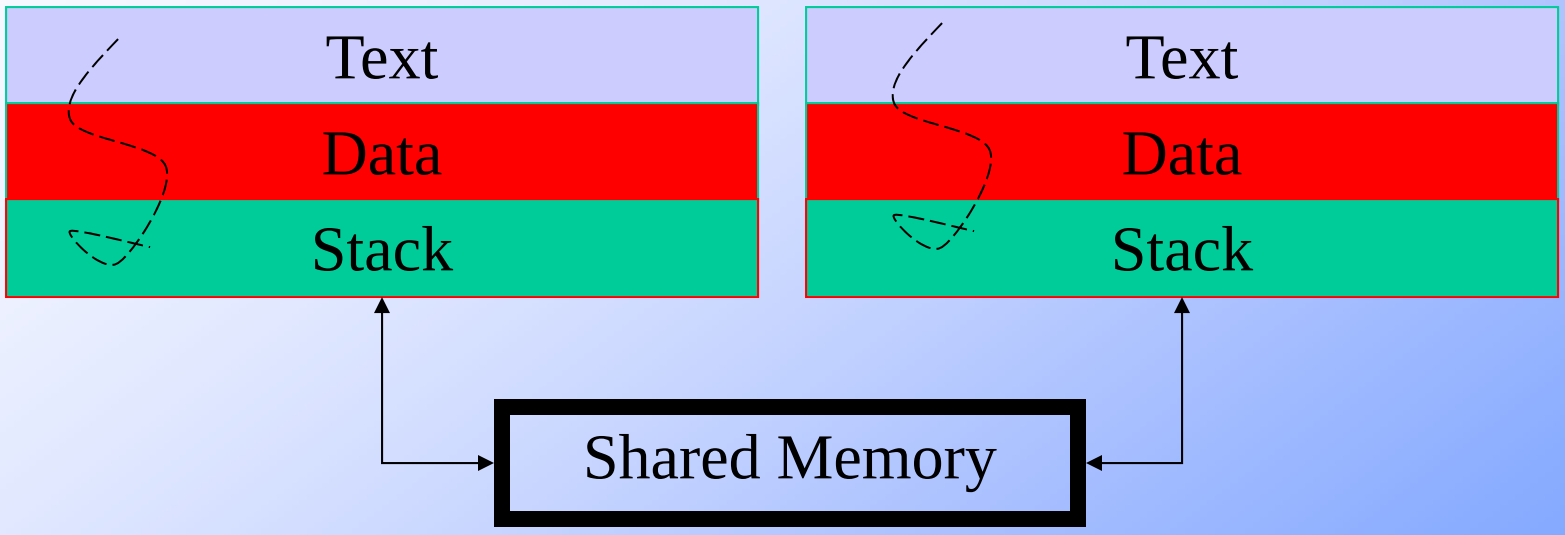
Storico

- I Threads non sono un invenzione di Java
  - Linguaggi
    - E.g. ADA
  - Sistemi Operativi
    - E.g. Windows
  - Esiste uno standard
    - P-threads (Posix Threads)
  - Java non segue nessuno di questi
    - Somiglia ai P-threads

Thread

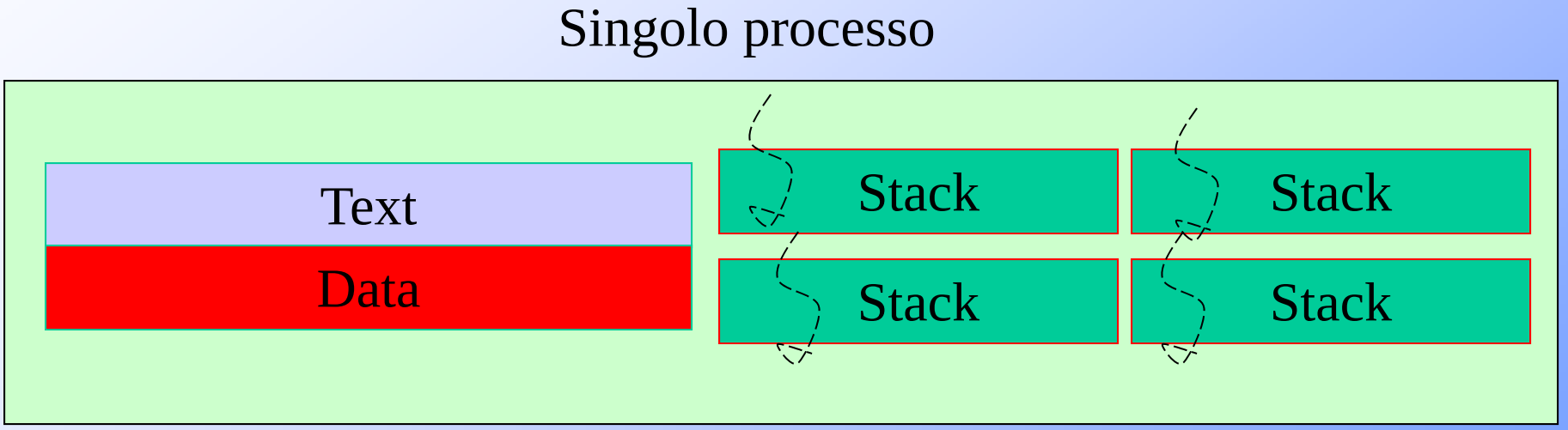
- Cos’e’ un thread?
- Thread == Thread of Control
- Il percorso di una specifica esecuzione di un programma
- La sequenza di operazioni eseguite in una specifica esecuzione di un programma

Segmenti di un programma: ambiente multitask, single thread



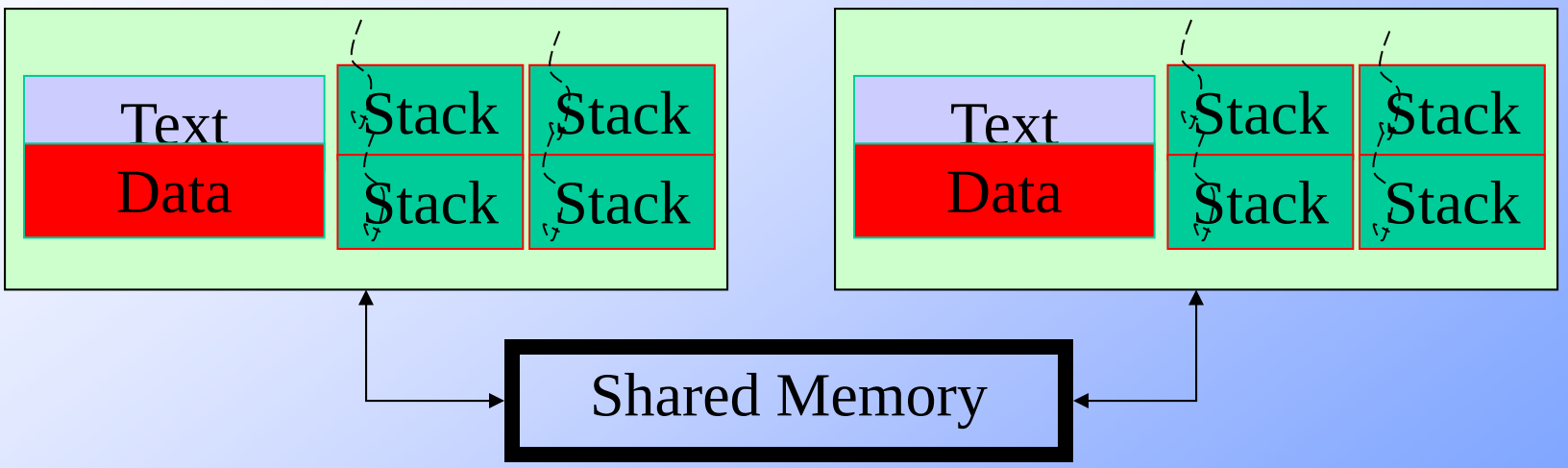
- Multitasking di diversi processi
  - Ogni processo un singolo thread
  - Area memoria condivisa per IPC (sync e.g. con semafori)

Segmenti di un programma: ambiente monotask, multi thread



- Un singolo processo
  - Un singolo segmento testo
  - Un singolo segmento dati(condivisi, ITC)
  - Uno stack per ogni thread of control

MultiTask, MultiThread



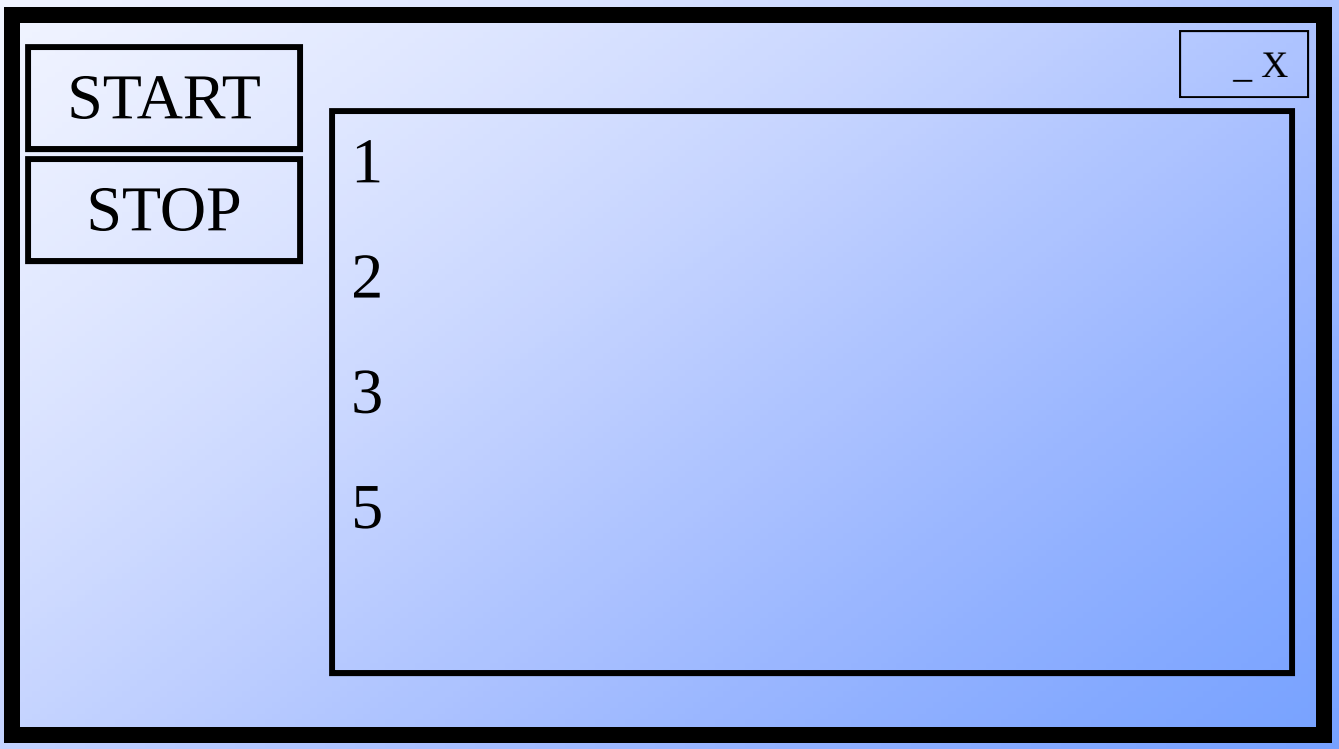
- Concorrenza a due livelli
  - ITC (intra process, inter thread)
  - IPC (inter process)

Perche’ Threads?

- Utilizzare ITC dove prima IPC
  - In generale un approccio “specialistico” e marginale
- Killer App? (beh, circa..)
- GUI
  - La gestione degli elementi di controllo
  - La gestione della “business logic”

Esempio 1: GUI

- Applicazione crivello di Eratostene



Soluzione monothread

```
stateStopped() {
    If(start.isPressed()) {
        Running = true;
        Return; }
    If(x.isPressed())
        Exit();
    If(fullScreen.isPressed())
        <azione grafica>
    If(iconify.isPressed())
        <azione grafica>
}
```

Soluzione monothread (cont.)

```
stateRunning() {
    <calcola prossimo primo>
     If(stop.isPressed()) {
          Running = false;
          Return; }
     If(x.isPressed())
          Exit();
     If(fullScreen.isPressed())
          <azione grafica>
     If(iconify.isPressed())
          <azione grafica>
}
```

Soluzione monothread (cont.)

```
Boolean running = false;
Main {
    while(true) {
     If(running)
          runningState();
     Else
          stoppedState();
     }
}
```

Soluzione Multithread

Thread 1

```
shouldRun = false;
While(true)
{
        <waitForEvent>
        If(start.isPressed()) {
                shouldRun = true;
                <startThread2> }
        If(stop.isPressed())
                shouldRun = false;
        If(x.isPressed())
                Exit();
        If(fullScreen.isPressed())
                <azione grafica>
        If(iconify.isPressed())
                <azione grafica>
}
```

Soluzione multithread (cont.)

Thread 2

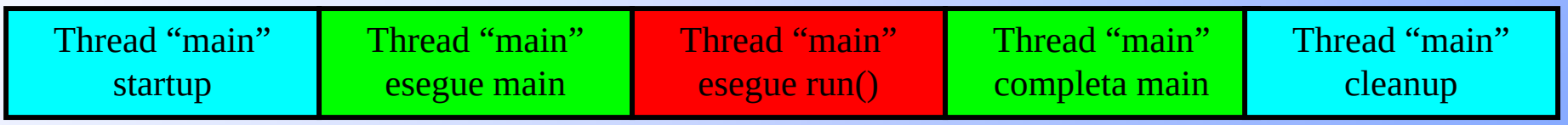
```
While(shouldRun) {
    <calcola prossimo primo>
    <forza ciclo di schedulazione>
}
```

Java Threads API

```
public class MyThread
{
        public void run()
        {
                int i = 0;
                for(i=0;i<100;i++)
                {
                        System.out.println("i = "
                          + i);
                }
        }
}
```

```
public class MyMain
{
        public static void main(String[]
argv)
        {
                (new MyThread()).run();
        }
}
```

Tempi di esecuzione



- Singolo flusso di esecuzione sequenziale

Nuova Versione

```
public class MyThread extends Thread
{
        public void run()
        {
                int i = 0;
                for(i=0;i<100;i++)
                {
                        System.out.println("i = "
                          + i);
                }
        }
}
```

```
public class MyMain
{
        public static void main(String[] argv)
        {
                (new MyThread()).start();
                int i = 0;
                for(i=0;i<100;i++)
                {
                        System.out.println
                        ("i = " + i);
                }
        }
}
```

Tempi di esecuzione

- Singolo flusso di esecuzione sequenziale sino alla “fork”
- Doppio flusso di controllo
  - Esecuzione concorrente di Thread “main” e “nuovo”
- Esecuzione sequenziale di cleanup

Thread Sleeping

sleep

```
public static void sleep(long millis)
                  throws InterruptedException
```

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.

Parameters:

`millis` - the length of time to sleep in milliseconds.

Throws:

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also:

`Object.notify()`

Thread Sleeping

sleep

```
public static void sleep(long millis,
                         int nanos)
                  throws InterruptedException
```

Causes the currently executing thread to sleep (cease execution) for the specified number of milliseconds plus the specified number of nanoseconds. The thread does not lose ownership of any monitors.

Parameters:

`millis` - the length of time to sleep in milliseconds.

`nanos` - 0-999999 additional nanoseconds to sleep.

Throws:

`IllegalArgumentException` - if the value of millis is negative or the value of nanos is not in the range 0-999999.

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also:

`Object.notify()`

Uso di sleep: un timer

```
public class MyTimer extends Thread
{
	long prev;
	long next;
	long interval;
	public static void main(String[] argv)
	{
		long i;
		if(argv.length > 0)
			i = Long.parseLong(argv[0]);
		else
			i = 1000;
		(new MyTimer(i)).start();
	}
```

un timer (cont.)

```
    public void run()
    {
        long current;
        prev = System.currentTimeMillis() - interval;
        while(true)
        {
            current = System.currentTimeMillis();
            System.out.println(current-prev);
            next = interval-( (current-prev)-interval);
            prev = current;
            try
                Thread.sleep(next);
            catch(InterruptedException ie)
                ie.printStackTrace();
        }
    }
}
```

Chi Vuol Esser Milionario?

- Perche' una signature cosi'
- Piu' precisamente:
  - PERCHE' STATIC?

Eredita': non e' rischioso giocarsela con la classe Thread?

- Puo' esserlo
  - Non posso ereditare da altro
- Soluzioni alternative?
  - Interfaccia Runnable

```
public interface Runnable
{
   public void run();
}
```

  - costruttore classe Thread con parametro

```
public Thread(Runnable r) { ...}
```

Revisione MyTimer

```
public class MyTimer Implements Runnable
{
        long prev;
        long next;
        long interval;
        public static void main(String[] argv)
        {
                long i;
                <come prima>
                Runnable r = new MyTimer(i);
                (new Thread(r)).start();
        }
        <etc.>
```

Cosa Succede in realta’

- metodo run di classe Thread

```
public void run()
{
      if(target != null)
            target.run();
}
```

- Se facciamo Thread by inheritance ⇒ overwrite di run
- Se facciamo Thread by interface ⇒ implementazione di default di metodo run delega all’oggetto target

Thread Join

- Attesa terminazione Thread

join

```
public final void join(long millis)
                  throws InterruptedException
```

Waits at most `millis` milliseconds for this thread to die. A timeout of `0` means to wait forever.

Parameters:

`millis` - the time to wait in milliseconds.

Throws:

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

Join (v2)

join

```
public final void join(long millis,
                       int nanos)
                throws InterruptedException
```

Waits at most `millis` milliseconds plus `nanos` nanoseconds for this thread to die.

Parameters:

`millis` - the time to wait in milliseconds.

`nanos` - 0-999999 additional nanoseconds to wait.

Throws:

`IllegalArgumentException` - if the value of millis is negative the value of nanos is not in the range 0-999999.

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

Join (v3)

join

```
public final void join()
                  throws InterruptedException
```

Waits for this thread to die.

Throws:

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

interrupt

`public void interrupt()`

Interrupts this thread.

First the `checkAccess` method of this thread is invoked, which may cause a `SecurityException` to be thrown.

If this thread is blocked in an invocation of the `wait()`, `wait(long)`, or `wait(long, int)` methods of the `Object` class, or of the `join()`, `join(long)`, `join(long, int)`, `sleep(long)`, or `sleep(long, int)`, methods of this class, then its interrupt status will be cleared and it will receive an `InterruptedException`.

If this thread is blocked in an I/O operation upon an interruptible channel then the channel will be closed, the thread's interrupt status will be set, and the thread will receive a `ClosedByInterruptException`.

If this thread is blocked in a `Selector` then the thread's interrupt status will be set and it will return immediately from the selection operation, possibly with a non-zero value, just as if the selector's `wakeup` method were invoked.

If none of the previous conditions hold then this thread's interrupt status will be set.

Throws:

`SecurityException` - if the current thread cannot modify this thread