

Sistema di Monitoraggio Ambientale

Simulazione multi-thread con rilevamento dati e gestione alert

Argomenti	Thread, lock, volatile, join, race condition, statistiche
Durata simulazione	60 secondi
Linguaggio	C# (.NET Framework / .NET Core)
Modalità	Individuale

1. Contesto del problema

Il Comune di Varese ha commissionato lo sviluppo di un sistema software per il monitoraggio in tempo reale della qualità dell'aria e del rumore urbano. Cinque centraline elettroniche, installate in punti strategici della città, rilevano periodicamente cinque parametri ambientali e trasmettono i dati a una centrale di controllo.

Il vostro compito è simulare questo sistema in C# usando i thread: ogni centralina è rappresentata da un thread indipendente che genera dati a intervalli regolari, li registra in una struttura condivisa e segnala eventuali superamenti di soglia.

2. Le centraline di rilevamento

Le cinque centraline sono installate nei seguenti punti della città. L'intervallo di rilevamento è proporzionale all'intensità del traffico nella zona: le centraline in aree ad alto traffico raccolgono dati più frequentemente.

ID	Posizione	Indirizzo	Intervallo	Traffico
C1	Piazza Repubblica	P.za Repubblica 1	3 secondi	Intenso
C2	Viale Borri	Viale Borri 120	5 secondi	Medio
C3	Via Caracciolo	Via Caracciolo 7	4 secondi	Medio-alto
C4	Parco Mirabello	Via Sanvito Silvestro	8 secondi	Basso
C5	Stazione FS	P.za Trento 1	6 secondi	Medio

3. Parametri rilevati e soglie di alert

Ogni centralina misura cinque parametri ambientali. Per ciascuno è definita una soglia massima: se un valore la supera, il sistema deve generare immediatamente un alert visibile sulla console.

Parametro	Unità di misura	Soglia alert	Descrizione
PM10	µg/m³	> 50	Particolato fine (polveri sospese)
PM2.5	µg/m³	> 25	Particolato ultrafine
CO2	ppm	> 1000	Biossido di carbonio
NO2	µg/m³	> 40	Biossido di azoto
Rumore	dB	> 70	Livello sonoro ambientale

4. Requisiti del programma

4.1 Strutture dati

Definire le seguenti classi:

- Rilevazione — contiene: IdCentralina (string), Timestamp (DateTime), e i cinque valori numerici (double).
- Centralina — contiene: Id, Posizione, Indirizzo (string) e Intervallo in secondi (int).

NOTA

Le classi devono essere semplici contenitori di dati (nessuna logica interna).
I valori numerici usano double per permettere decimali (es. PM10 = 34.7 µg/m³).

4.2 Thread e ciclo di rilevamento

Per ogni centralina va creato un Thread separato che esegue il seguente ciclo fino al termine della simulazione:

1. Generare una rilevazione con valori casuali realistici (vedere Tabella Valori di riferimento pag. 4).
2. Aggiungere la rilevazione alla lista condivisa — proteggere l'accesso con lock.
3. Controllare se uno o più parametri superano la soglia — in caso affermativo stampare un alert colorato in rosso sulla console (vedere Alert pag.4).
4. Attendere l'intervallo specifico di quella centralina con Thread.Sleep().

4.3 Risorse condivise e uso del lock

Il programma usa le seguenti risorse condivise tra thread, che richiedono protezione:

Risorsa	Tipo	Perché va protetta
Lista rilevazioni	List<Rilevazione>	List<T> non è thread-safe: due Add() simultanei corrompono la struttura interna.
Generatore casuale	Random	Random non è thread-safe: due chiamate simultanee producono risultati errati.
Console	Console.Write/WriteLine	Senza lock, due thread stampano contemporaneamente mescolando le righe.

Usare un unico oggetto lock per tutte e tre le risorse: `private static readonly object _lock = new object();`

4.4 Gestione degli alert

Al termine di ogni rilevazione, confrontare ogni parametro con la propria soglia. Se almeno uno la supera:

- Raccogliere in una lista i nomi dei parametri fuori soglia con il valore misurato e la soglia.
- Stampare sulla console, all'interno del lock, un messaggio di alert in rosso (ConsoleColor.Red) nel formato:

```
⚠ ALERT [12:34:05] C1 - Piazza Repubblica (P.za Repubblica 1)
  → PM10 = 67.3 µg/m³ (soglia 50) SUPERATA
  → NO2 = 48.1 µg/m³ (soglia 40) SUPERATA
```

Alert

- Se nessun parametro supera la soglia, stampare la riga di rilevazione normale in verde (ConsoleColor.Green).

- Ripristinare sempre il colore originale della console dopo ogni stampa (salvare `ConsoleColor.ForegroundColor` prima di modificarlo).

4.5 Avvio e terminazione

Il thread principale (Main) deve:

5. Creare e avviare i 5 thread, uno per centralina.
6. Attendere 60 secondi con `Thread.Sleep(60000)`.
7. Calcolare e stampare il report finale.

4.6 Report finale

Al termine della simulazione stampare un riepilogo per ogni centralina. Il report deve contenere obbligatoriamente:

Dati quantitativi

- Numero totale di rilevazioni effettuate dalla centralina.
- Numero atteso di rilevazioni (= 60 / intervallo) — confrontare con quello effettivo.
- Per ogni parametro: valore minimo, massimo e media aritmetica registrati durante la simulazione.
- Numero totale di alert generati dalla centralina.
- Numero di alert per ogni singolo parametro.

Calcolo delle statistiche

Le statistiche vanno calcolate iterando sulla lista filtrata per `IdCentralina`. Per ogni parametro:

- Media: sommare tutti i valori e dividere per il conteggio.
- Minimo e massimo: scorrere la lista tenendo traccia del valore più basso e più alto incontrati.
- Alert: contare le rilevazioni in cui il valore supera la soglia.

NOTA

Le statistiche vengono calcolate DOPO `Thread.Join()` su tutti i thread, quindi la lista è definitiva e non serve il lock per leggerla.

Formato del report

```
===== REPORT FINALE =====
Rilevazioni totali: 57

--- C1 | Piazza Repubblica | P.za Repubblica 1 ---
Rilevazioni: 20 (attese: 20, ogni 3s)
PM10   : min= 4.2  max=73.1  media=28.4  alert=2
PM2.5  : min= 1.1  max=31.6  media=12.7  alert=1
CO2    : min=412   max=1143  media=698   alert=3
NO2    : min= 3.4  max=52.8  media=19.2  alert=1
Rumore : min=41.2  max=84.5  media=59.3  alert=2
Alert totali: 9
```

5. Generazione dei valori simulati

Poiché le centraline sono simulate, i valori vengono generati casualmente con `Random`. Per rendere la simulazione realistica applicare queste regole:

- Con probabilità del 90%: generare un valore nel range normale (da 0 alla soglia).
- Con probabilità del 10%: generare un valore oltre la soglia (picco di inquinamento).

Parametro	Range normale (90%)	Range picco (10%)	Soglia
PM10	0 – 50 µg/m³	50 – 80 µg/m³	50
PM2.5	0 – 25 µg/m³	25 – 40 µg/m³	25
CO2	400 – 1000 ppm	1000 – 1200 ppm	1000
NO2	0 – 40 µg/m³	40 – 60 µg/m³	40
Rumore	40 – 70 dB	70 – 90 dB	70

Tabella Valori di riferimento

ATTENZIONE

Random non è thread-safe. Usare un'istanza condivisa protetta con lock, oppure creare un'istanza locale per ogni thread con un seme diverso (es. basato sull'ID centralina).

6. Concetti da conoscere e applicare

Thread

Un thread è un flusso di esecuzione indipendente all'interno dello stesso processo. Nel programma ogni centralina gira su un thread separato: i 5 thread e il thread principale girano in parallelo sullo stesso processo, condividendo memoria e risorse.

- `new Thread(metodo)` — crea il thread.
- `thread.Start()` — avvia l'esecuzione.
- `thread.IsBackground = true` — il thread viene terminato automaticamente se il processo principale termina.
- `thread.Join()` — blocca il chiamante finché il thread non termina.

Lock e race condition

Una race condition si verifica quando due thread accedono contemporaneamente a una risorsa condivisa e il risultato dipende dall'ordine casuale delle esecuzioni. Il lock risolve il problema garantendo accesso esclusivo:

- Solo un thread alla volta può essere dentro il blocco lock.
- Gli altri thread che raggiungono lo stesso lock si mettono in coda e aspettano.
- L'oggetto passato a lock deve essere readonly e non deve mai essere null.

volatile

La keyword volatile impedisce al compilatore e alla CPU di ottimizzare l'accesso a una variabile che può essere modificata da un thread diverso. Senza volatile, un thread potrebbe leggere una copia locale (cache) della variabile e non vedere mai la modifica.

Thread.Sleep vs Thread.Join

	Thread.Sleep(ms)	Thread.Join()
Chi si blocca	Il thread che chiama Sleep	Il thread che chiama Join
Per quanto	Per il numero di millisecondi specificato	Finché il thread target non termina
Uso nel progetto	Ogni centralina aspetta il proprio intervallo tra una rilevazione e l'altra	Main aspetta che tutti i thread abbiano finito prima del report