Sistemi Operativi

Progetto sessione estiva 2020/2021

Relazione del progetto

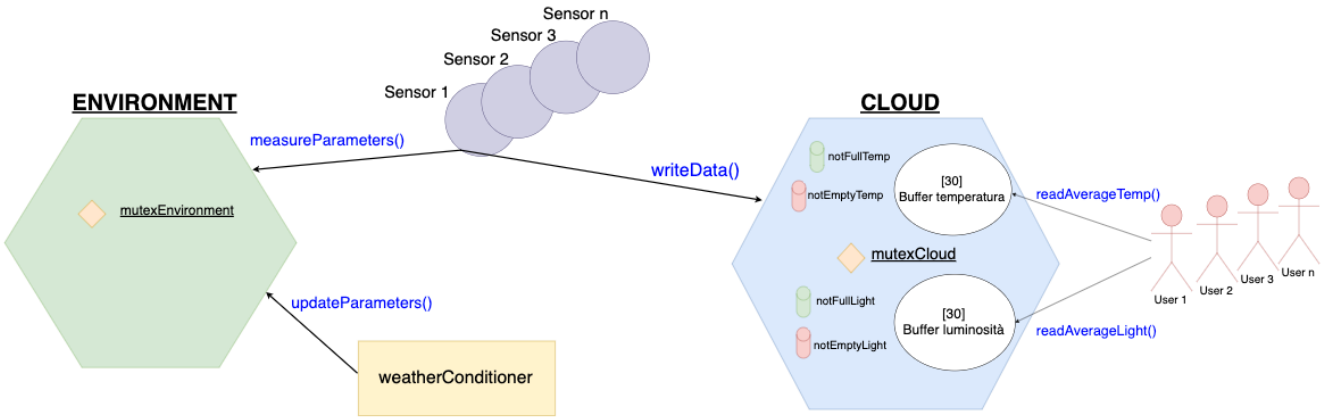
Dalia Abbruciati

Matricola 277914

1) Specifica del problema

Scrivere un programma multithread che simuli il funzionamento di un sistema Internet of Things (IoT) per la raccolta di informazioni ambientali e l'aggregazione su cloud. In particolare, il sistema sarà costituito da un numero *nSensors* di sensori in grado di accedere all'ambiente e misurare temperatura e luminosità. Le condizioni ambientali saranno controllate da un thread (*WeatherConditioner*) che avrà il compito di modificare periodicamente i valori di temperatura e luminosità presenti nell'ambiente secondo la relazione descritta in seguito. I sensori, dopo aver effettuato la misurazione, invieranno i dati in cloud dove saranno memorizzati utilizzando dei buffer circolari a capacità prefissata agendo, in questo modo, come dei produttori. Saranno, inoltre, presenti *nUsers* utenti che sottometteranno richieste per ottenere i valori medi di temperatura e luminosità registrati in cloud. Ogni lettura da parte di un utente sarà ottenuta mediando 4 valori cariati nel buffer in ordine FIFO. La lettura comporterà la rimozione dei 4 valori letti come avviene per un normale consumatore. Avendo il buffer una capacità limitata i consumatori dovranno attendere la presenza di dati nel buffer ed i produttori dovranno attendere in caso di mancanza di spazio. I sensori in attesa di scrivere e gli utenti in attesa di leggere saranno gestiti in ordine FIFO.
La simulazione terminerà quando tutti gli utenti avranno effettuato 100 letture ciascuno.

2) Descrizione della progettazione



Come si può osservare dal grafico, in questa specifica ci sono due oggetti condivisi, *Environment* e *Cloud*, identificati dai due esagoni. I thread *Sensor* misurano i dati di temperatura e luminosità dall'*Environment* i quali verranno poi scritti negli appositi array di buffer circolari, contenuti nell'oggetto *Cloud*. Quando in entrambi i buffer saranno presenti 4 o più valori, i semafori contatori notEmpty diventeranno verdi e i thread User potranno leggere e rimuovere i dati dai buffer.

2.1) Scelte di progetto

Per quanto riguarda gli attributi di sincronizzazione:

- Semafori contatori: ho scelto di usare 2 semafori contatori in ogni buffer per gestire al meglio l'acquisizione e il rilascio dei permessi e per garantire la politica FIFO; il semaforo *notFull* notifica ai *Sensor* che il buffer non ha ancora raggiunto la sua capacità massima e possono inserire nuovi dati mentre *notEmpty* sveglia gli *User* quando saranno disponibili almeno 4 dati da leggere;
- Semafori binari: ho usato il ReentrantLock negli oggetti condivisi per garantire la mutua esclusione nei metodi condivisi da più thread che operano sulle stesse variabili.

3) Descrizione dell'implementazione

Di seguito andrò a descrivere le varie classi che compongono il programma:

● **Classe Environment**

Environment è l'oggetto condiviso dai thread *Sensor* e *WeatherConditioner,* che al suo interno contiene:

- una struttura dati di tipo array per memorizzare i dati di tipo temperatura e luminosità;
- 2 variabili double per memorizzare i valori misurati di temperatura e luminosità;
- semaforo binario ReentrantLock per garantire la mutua esclusione sui metodi *measureParameters()* e *updateParameters()* a guardia delle variabili condivise di temperatura e luminosità.

Per quanto riguarda i metodi della classe:

- **measureParameters(Sensor s):** metodo invocato dal sensore in cui vengono salvati i dati dei valori misurati nell'array apposito formato da due celle dove nella prima vengono salvati i dati di temperatura e nella seconda quelli di tipo luminosità.

- **updateParameters():** metodo invocato dal thread *WeatherConditioner* che semplicemente applica la formula per l'aggiornamento dei parametri richiesta dalla specifica del progetto. Esso è in mutua esclusione per far sì che nel momento dell'aggiornamento i valori aggiornati siano effettivamente quelli misurati dai sensori.

In entrambi i metodi le eccezioni vengono catturate direttamente nelle rispettive classi *Sensor* e *WeatherConditioner*.

● **Classe Cloud**

Cloud è l'oggetto condiviso tra i thread *Sensor* e *Users*. I suoi attributi interni funzionali comprendono:

- 2 array di tipo double che rappresentano *bufferTemp* e *bufferLight* per contenere i dati;
- 2 contatori degli elementi nei buffer;
- 4 puntatori logici di input e output, due per ogni buffer, per aggiornare i dati.

Per quanto riguarda invece gli attributi di sincronizzazione:

- 1 ReentrantLock per garantire la mutua esclusione tra i metodi di scrittura dati, invocato dai sensori, e di lettura e rimozione dei dati invocati dagli utenti, a guardia delle loro variabili condivise;
- 2 semafori contatori, *notFullTemp* e *notEmptyTemp*, per gestire la sospensione o il risveglio dei thread *Sensor* e *User* sul buffer temperatura;
- 2 semafori contatori, *notFullLight* e *notEmptyLight*, per gestire la sospensione o il risveglio dei thread *Sensor* e *User* sul buffer luminosità;
- 2 contatori dei permessi dei buffer.

Per quanto riguarda i metodi della classe:

- **writeData(Sensor s, double dataTemp, double dataLight):** metodo bloccante invocato dal sensore per scrivere i dati all'interno dei due buffer. Se i 2 buffer sono pieni, il sensore si sospende attendendo che ci sia almeno uno spazio libero in entrambi per scrivere i dati, altrimenti li inserisce aggiornando di volta in volta il puntatore logico di input. Appena nei buffer sono presenti contemporaneamente 4 valori da leggere, sveglio un utente in ordine FIFO e decremento il contatore dei permessi;
- **readAverageTemp(User u):** metodo bloccante invocato dall'utente per leggere i dati nel buffer temperatura. Se non sono presenti almeno 4 dati da leggere in entrambi i buffer, l'utente rimane sospeso in attesa, altrimenti legge i 4 valori, fa la media di essi tramite il metodo *getAvgDataTemp()* e aggiorna il puntatore logico di output. Infine, notifica al *Sensor* che ci sono 4 nuovi posti liberi disponibile nel buffer facendo una release di 4 permessi;
- **readAverageLight(User u):** stesso metodo descritto in precedenza ma il buffer considerato è quello dei dati della luminosità.
- **getAvgDataTemp(int out):** metodo per ricavare il valore medio dei 4 dati inseriti e letti dagli utenti nel buffer temperatura.
- **getAvgDataLight(int out):** metodo per ricavare il valore medio dei 4 dati inseriti e letti dagli utenti nel buffer luminosità.

● **Classe Sensor**

Contiene i metodi:

- **run():** metodo che implementa il comportamento del thread e che prevede l'esecuzione dei seguenti passi fino alla fine della simulazione:
  1) misura I parametri ambientali invocando il metodo *measureParameters()* dell'oggetto Environent;
  2) applica la percentuale di errore;
  3) invia i dati al *Cloud* invocando il metodo *writeData()*;
  4) si sospende per 400 millisecondi.
- **setErrore(int errore, double lettura):** metodo che applica l'errore generato all'inizio della creazione del thead ai valori misurati nell'ambiente. Dopo aver ricavato il valore dell'errore generato casualmente nell'intervallo [-10,10], se esso è positivo lo sommo al valore letto, oppure, se esso è negativo lo sottraggo al valore della lettura.

● **Classe User**

Contiene i metodi:

- **run():** implementa il comportamento del thread che eseguirà i seguenti passi per 100 volte:
  1) attende un tempo casuale estratto nell'intervallo [0,99] millisecondi;
  2) salvo il tempo tramite *System.currentTimeMillis()* prima di effettuare le letture dei dati;
  3) letto i dati nei buffer temperatura e luminosità;
  4) salvo il tempo dopo aver effettuato le letture;
  5) calcolo il tempo trascorso dall'inizio fino alla fine della lettura;
  6) sommo i tempi di lettura di un singolo utente fatti per tutta la durata della smulazione.
- **getAvgTime():** metodo che calcola la media del tempo di lettura di un singolo thread Utente;
- **getSommaTempoLetturaTot():** metodo che restituisce il valore della somma del tempo di lettura totale di un singolo utente.

● **Classe Weather conditioner**

Contiene il metodo:

- **run():** implementa il comportamento del thread che dovrà eseguire i seguenti passi per tutta la durata della simulazione:
  1) si sospende di 400 millisecondi;
  2) aggiorna i valori di temperatura e luminosità secondo le seguenti relazioni:
     - *Luminosità(t1) = Luminosità(t0) + 1000; con luminosità iniziale = 0*
     - *Temperatura(t1) = 10+0.00022* Luminosità(t1)*

In tutte le classi è stato implementato anche il metodo ***getTempoTrascorso()*** che calcola il tempo trascorso dall'inizio della simulazione.
Nella classe **Main**, dopo il termine della simulazione, sono stati implementati i metodi che stampano la media e la deviazione standard del tempo necessario per effettuare la lettura dei valori di temperatura e luminosità da parte dei singoli *Users* scelti tramite riga di comando dall'utente che esegue la simulazione. Per quanto riguarda la:

- media: viene calcolata la somma per ogni lettura effettuata da ogni utente tramite il metodo ***getSommaTempoLetturaTot()***, infine per ottenere la media divido il valore ottenuto per il numero di *nUsers* moltiplicato per il numero di letture effettuato da ognuno di essi;
- deviazione standard: il tempo di ogni lettura effettuata dall'utente viene sottratto al valore della media, ricavata in precedenza, per ottenere il valore dello scarto. Calcolo la varianza, o scarto quadratico, elevando al quadrato il valore dello scarto e infine ricavo la deviazione standard facendo la radice quadrata della varianza fratto il numero di *nUsers* per il numero di letture da loro effettuate.

Le parti di codice state usate per l'analisi sperimentale nel punto 3 della specifica d'esame sono state commentate.

4) Testing del programma

Input iniziali scelti dall'utente:

```
run:
Inserire il numero di sensori:
10
Inserire il numero di utenti:
5
```

Output che indica i tempi medi di lettura per ogni *User* scelto dall'utente:

```
--- SIMULAZIONE TERMINATA ---

--> Tempo medio di lettura di Utente_0: 738.09ms
--> Tempo medio di lettura di Utente_1: 751.05ms
--> Tempo medio di lettura di Utente_2: 747.6ms
--> Tempo medio di lettura di Utente_3: 751.49ms
--> Tempo medio di lettura di Utente_4: 746.13ms
```

Output che rappresenta il tempo medio e la deviazione standard **totale** di lettura da parte di tutti gli *User* :

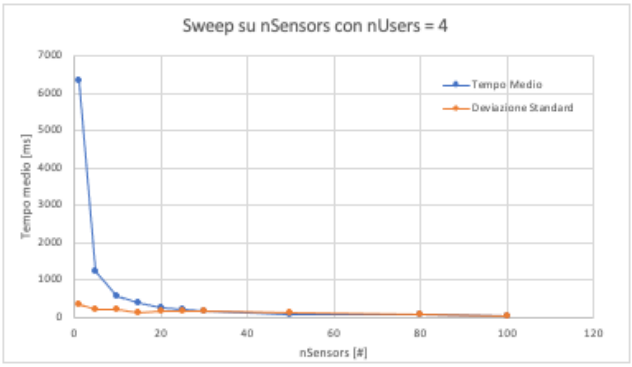
```
TEMPO MEDIO TOTALE DELLE LETTURE: 746,87ms
DEVIAZIONE STANDARD TOTALE: 67,38ms
```

4.1) Analisi sperimentale

- Andamento del tempo medio e deviazione standard da parte degli *nUsers* al variare del numero di *nSensors:*

Sweep nSensors nUsers = 4

	Tempo Medio	Deviazione Standard
1	6358,54	334,03
5	1222,98	189,69
10	570,53	196,52
15	367,04	105,96
20	264,37	141,36
25	196,09	155,31
30	157,89	153,2
50	74,73	112,65
80	54,44	56,82
100	48,29	45,73



Osservando il grafico possiamo notare che all'aumentare del numero di *nSensors*, il tempo medio e la deviazione diminuiscono sempre di più, in linea con i risultati aspettati, poiché aumentando il numero di sensori che scrivono nei buffer, essi conterranno sempre elementi da leggere in modo tale da minimizzare il tempo di attesa per le letture dei dati da parte degli *nUsers*. Il tempo medio ha una fase di decrescita più rapida delle altre quando si passa da 1 a 5 *nSensors* per poi decresce linearmente; la deviazione invece ha un andamento lineare decrescente abbastanza omogeneo.

- Andamento del tempo medio e deviazione standard da parte degli *User* al variare del numero di *nUsers:*

Sweep nUsers nSensor = 10

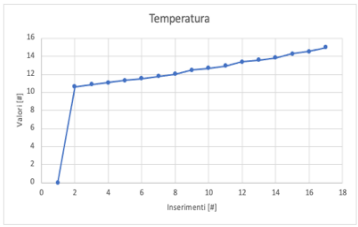
	Tempo Medio	Deviazione Standard
1	109,5	139,85
5	748,29	67,05
10	1547,61	108,1
15	2352,35	145,77
20	3151,12	200,45
25	3951,4	245,08
30	4752,73	289,27
50	7955,31	475,53
80	12801,91	760,12
100	16234,49	2541,33

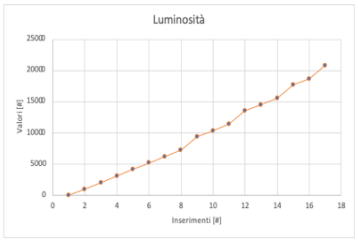
In questo grafico possiamo dedurre che all'aumentare del numero di *nUsers*, aumentano anche il tempo medio e la deviazione delle letture effettuate. Il tempo medio ha un andamento lineare crescente, mentre la deviazione standard è anch'essa lineare ma con una crescita più evidente quando si passa da 80 a 100 *User*. Invece nel caso con 10 sensori e 1 utente notiamo che la deviazione ha un valore maggiore rispetto al tempo medio misurato, mentre in tutti gli altri casi è sempre inferiore.

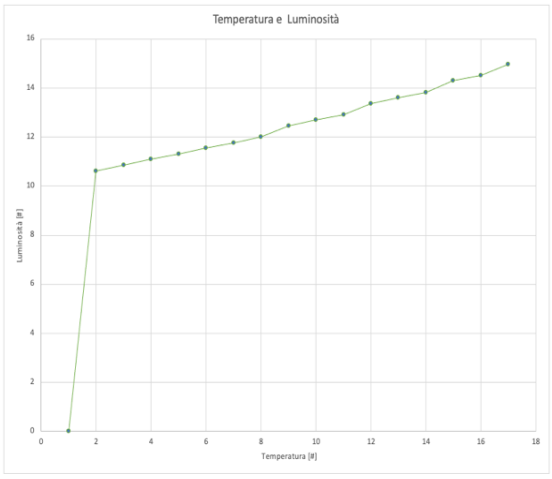
- Relazione tra valori di temperatura e luminosità misurati dal sensore numero 3, nel caso con **nSensors = 100** e **nUsers = 4**

Sensore [3] nSensors = 100 nUsers = 4
errore 4%

Numero inserimenti	Temperatura	Luminosità
1	0.0	0.0
2	10,6288	1040
3	10,8576	2080
4	11,0864	3120
5	11,3152	4160
6	11,544	5200
7	11,7728	6240
8	12,0016	7280
9	12,4592	9360
10	12,688	10400
11	12,9168	11440
12	13,3744	13520
13	13,6032	14560
14	13,832	15600
15	14,2896	17680
16	14,5184	18720
17	14,976	20800





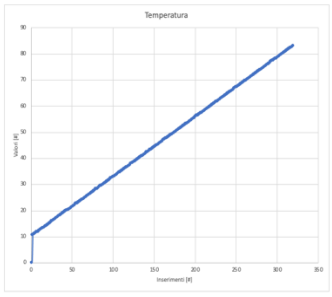


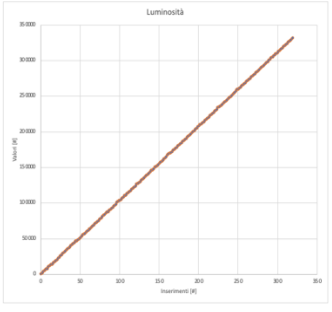
In questo primo caso possiamo notare dal grafico che la temperatura ha una fase iniziale in cui cresce rapidamente per poi adottare un andamento lineare dei suoi valori, dati dalla formula espressa nella specifica, mentre la luminosità ha un andamento lineare crescente. Possiamo inoltre osservare dai valori evidenziati che essi, man mano che vengono inseriti nei buffer, subiscono dei "salti", questo perché avendo molti sensori che scrivono e il *weatherConditioner* che aggiorna periodicamente i valori, i buffer saranno quasi sempre saturi, anche a causa dei pochi *nUsers* che leggono i dati, perciò i sensori dovranno attendere uno o più turni prima di poter riuscire ad inserirei dati.

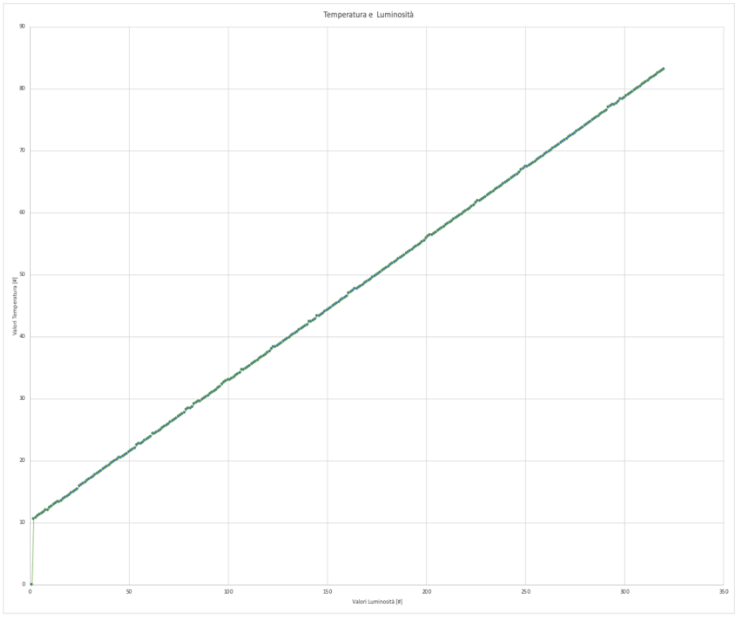
- Relazione tra valori di temperatura e luminosità misurati dal sensore numero 3, nel caso con **nSensors = 5 e nUsers = 4.**

Sensore [3] nSensors = 5 nUsers = 4
errore 4%

Numero inserimenti	Temperatura	Luminosità
1	0	0.0
2	10,6288	1040
3	10,8576	2080
4	11,0864	3120
5	11,3152	4160
6	11,544	5200
7	11,7728	6240
8	12,0016	7280
9	12,0016	7280
10	12,4592	9360
11	12,688	10400
12	12,9168	11440
13	13,1456	12480
14	13,3744	13520
15	13,3744	13520
16	13,6032	14560
17	13,832	15600
18	14,0608	16640
19	14,2896	17680
20	14,5184	18720
21	14,7472	19760
22	14,976	20800
23	15,2048	21840
24	15,4336	22880
25	15,8912	24960
26	16,12	26000
27	16,3488	27040
28	16,5776	28080
29	16,8064	29120
30	17,0352	30160
31	17,264	31200
32	17,4928	32240
33	17,7216	33280
34	17,9504	34320
35	18,1792	35360
36	18,408	36400
37	18,6368	37440
38	18,8656	38480
39	19,0944	39520
40	19,3232	40560
41	19,552	41600
42	19,7808	42640
43	20,0096	43680
44	20,2384	44720
45	20,4672	45760
46	20,4672	45760
47	20,696	46800
48	20,9248	47840
49	21,1536	48880
50	21,3824	49920
51	21,6112	50960
52	21,84	52000
53	22,0688	53040







In questo secondo caso, i valori misurati (in totale 320) di temperatura e luminosità hanno lo stesso andamento di quelli riportati nel caso precedente, ma avendo qui un numero inferiore di *nSensors* essi non rimangono in attesa di scrivere nel buffer perché esso sarà sempre liberato dagli *nUsers* e per questo, come possiamo notare dai valori in rosso, può capitare che il sensore legga due volte lo stesso dato perché esegue più velocemente del *weatherConditioner*.
In entrambi i casi, comunque, il rapporto tra i dati di temperatura e luminosità è proporzionale, ovvero variano entrambi nello stesso modo.