Sviluppo di un sondaggio sulla comprensione dei threads tra gli studenti delle scuole superiori

Emanuele Scapin e Nicola Dalla Pozza

Istituto Tecnico Tecnologico G. Chilesotti - 36016 Thiene (VI)

Introduzione e contesto

Premessa

Le difficoltà degli studenti delle scuole superiori nell'apprendere la programmazione concorrente sono ben note ai docenti di Informatica.

Nonostante ciò, possiamo osservare che a questo argomento non è stata posta adeguata attenzione nel contesto della formazione sulla programmazione.

Mentre a livello internazionale è ancora aperto il dibattito se tale argomento sia da inserire nei curricula pre-universitari, le linee guida ministeriali del nostro Paese per gli istituti tecnici con specializzazione in Informatica indicano che gli studenti sono tenuti ad acquisire competenze chiave di programmazione concorrente.

Questo lavoro vuole presentare come è stato organizzato il questionario proposto agli studenti per investigare le loro difficoltà con la programmazione concorrente e i thread.

Introduzione

L'indagine è rivolta a trovare alcune risposte preliminari alle seguenti domande di ricerca:

1. In che misura gli studenti sono a loro agio con i concetti di base della programmazione concorrente?
2. In che misura la percezione della fiducia in sé stessi da parte degli studenti è correlata alle loro prestazioni effettive in semplici compiti di programmazione concorrente?
3. Quali sono le maggiori difficoltà nell'apprendimento di questo paradigma?

Strumento d’indagine

Questionario

Complessivamente il sondaggio comprende:

24 domande, 11 delle quali basate su 4 piccoli compiti

7 domande sono di comprensione del programma

4 richiedono valutazioni della fiducia in sé stessi sulle risposte fornite in una scala Likert a 4 gradi.

Il sondaggio completo è reperibile al link
`https://forms.gle/W4HxuvCEDJyegjfh9`.

Struttura generale

1. Informazioni generali (2 domande)
genere; preferenza soggettiva dei linguaggi di programmazione usati con i thread

2. Concetti generali relativi ai thread (7 domande di percezione soggettiva)
difficoltà dell'argomento in generale; self-efficacy in generale; adeguatezza del tempo dedicato all'argomento; adeguatezza degli esempi proposti; difficoltà con concetti/problemi specifici relativi ai thread; difficoltà con specifici strumenti di programmazione relativi ai thread; difficoltà con la sincronizzazione dei thread

3. Task (7 domande di comprensione del programma e 4 sulla valutazione della self-confidence delle risposte fornite)
Task1 – mutua esclusione (4 domande); Task2 – schema produttore-consumatore; Task3 – analisi di sincronizzazione; Task4 – analisi di stallo (deadlock)

4. Possibili strumenti di aiuto (3 domande di percezione soggettiva)
se le rappresentazioni grafiche siano mai state prese in considerazione; potenziale atteso delle rappresentazioni grafiche; percezione dell'uso di specifiche rappresentazioni grafiche

5. Suggerimenti (1 domanda)
suggerimenti per rendere più interessanti e chiare le lezioni sui thread

Domande

Le domande che affrontano l'apprendimento dei concetti di concorrenza sono ispirate da Choi e Lewis[1], che infatti hanno catalogato gli errori che gli studenti tipicamente commettono quando si approcciano a programmi multi-thread, in particolare in relazione:

- race conditions,
- deadlock,
- problemi di sincronizzazione.

[1] Sung-Eun Choi and E. Christopher Lewis. A study of common pitfalls in simple multi-threaded programs. SIGCSE Bull., 32(1):325–329, 3 2000.

Problemi

I problemi inseriti nel sondaggio sono riportati di seguito. Per adattarsi alle comuni pratiche nelle scuole superiori coinvolte il codice di riferimento è simile a Java.

I primi due problemi sono stati ispirati da programmi proposti da Meyer e al. in "*Concurrent Programming with Java Threads*",[2] e hanno lo scopo di verificare la capacità degli studenti nell'anticipare i risultati di threads simultanei.

[2] Il materiale del corso è disponibile al link `https://se.inf.ethz.ch/courses/2011a_spring/soft_arch/lectures/old/13_softarch_self_study_threads.pdf`.

Problema 1

Il primo problema presenta una classe molto semplice volta a sincronizzare l'accesso ad una risorsa condivisa, in base alla disponibilità dei dati.

Sono state quindi presentate quattro sequenze temporali di invocazioni di metodi da parte di due thread concorrenti, con la richiesta di identificare gli esiti risultanti (domande a–d).

```
public class Counter {
private int count = -1; // a negative value of count is

public synchronized int getCount() {
    while ( count < 0 ) {
        try {
            wait();
        } catch ( Exception e ) {}
    }
    return count;
}

public synchronized void setCounter( int initValue ) {
    if ( initValue >= 0 ) {
        count = initValue;
        notify();
    }
}

public synchronized void increment() {
    while ( count < 0 ) {
        try {
            wait();
        } catch ( Exception e ) {}
    }
```

Problema 1

Analizza l'esecuzione dei frammenti di codice in Figura, relativi a due thread distinti, Thread-1 e Thread-2, che operano su un'istanza condivisa x della classe Counter introdotta sopra. Le operazioni di ciascuno dei due thread sono rappresentate lungo i lati opposti dell'asse verticale, secondo l'ordine temporale (dall'altro verso il basso) in cui i metodi invocati nelle istruzioni vengono avviati; inoltre, nessuna operazione su x oppure su i è stata omessa nei flussi riportati. Quali sono i valori stampati in output nel corso dell'esecuzione di Thread-1?

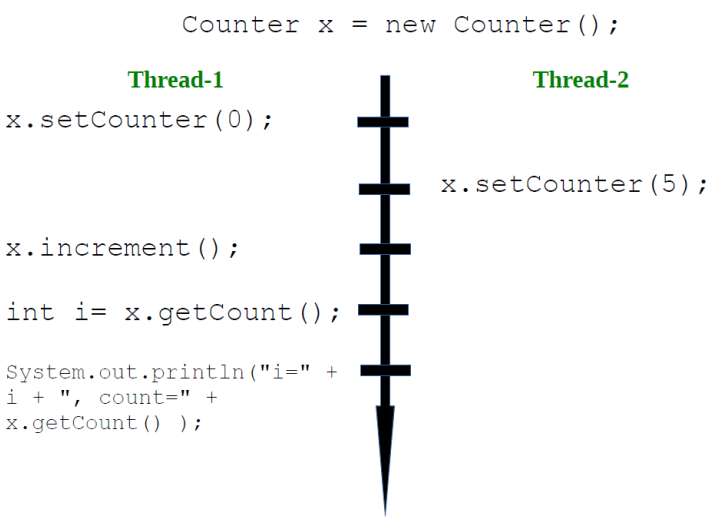

Figura: Problema 1.a

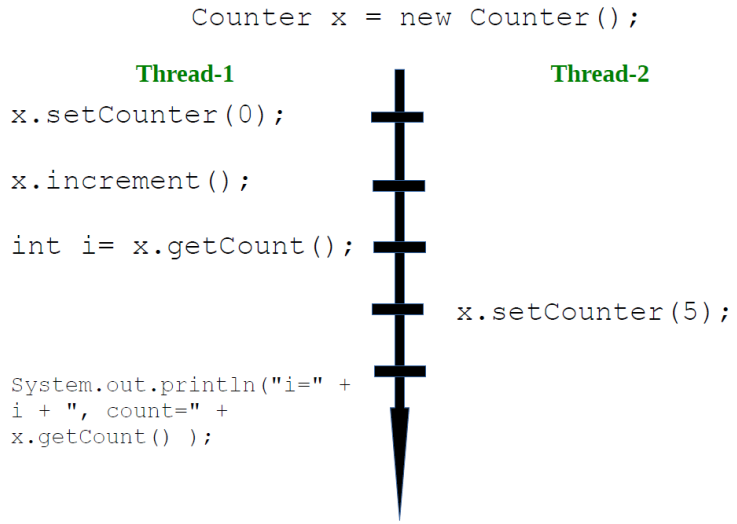

Figura: Problema 1.b

Problema 1

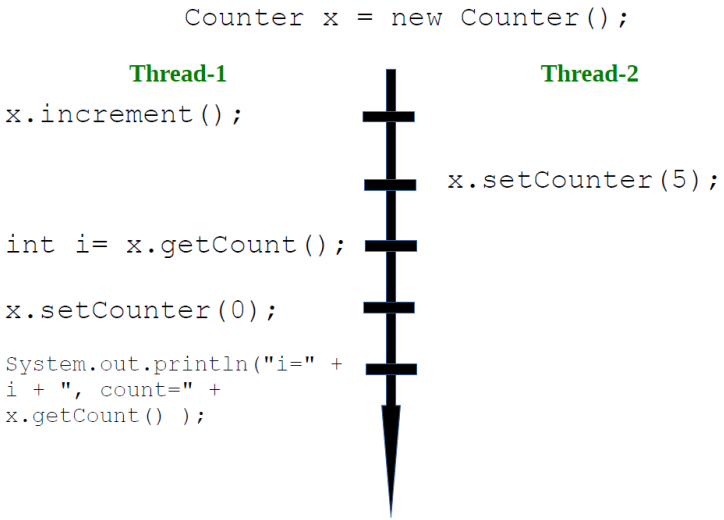

Figura: Problema 1.c

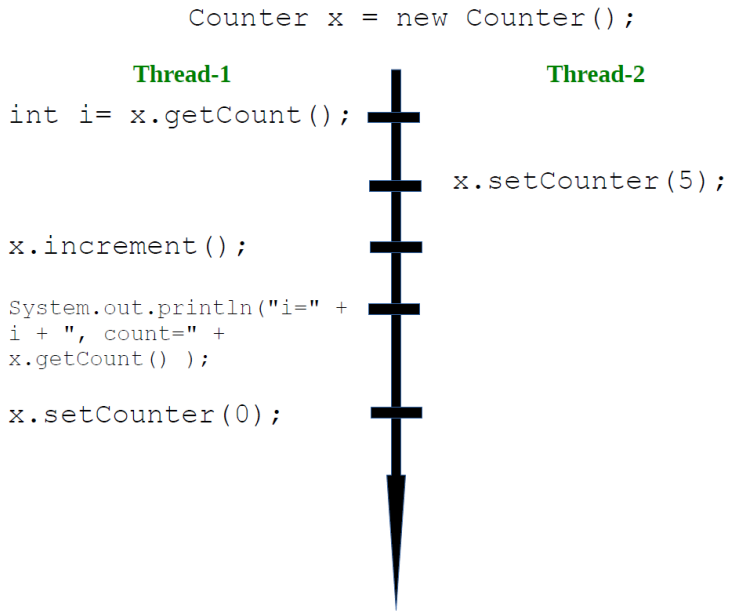

Figura: Problema 1.d

Problema 2

```
class Adder extends Thread {

  private int loops;
  private Vector<Integer> buffer;   // integer sequence

  public Adder( int loops, Vector<Integer> buffer ) {

    this.loops = loops;
    this.buffer = buffer;
  }

  public void run() {

    for ( int i=0; i<loops; i=i+1 ) {

      synchronized ( buffer ) {
        while ( buffer.size() < 2 ) {
          try {
            buffer.wait();
          } catch ( Exception e ) {}
        }
        // n: adds the first two buffer elements
        int n = buffer.get(0) + buffer.get(1);
        buffer.clear();   // buffer is emptied
        System.out.print( ""+n );
        buffer.notify();
    }}
    System.out.println();
  }

}  // Adder
```

```
class Provider extends Thread {

  private int[] stream;
  private Vector<Integer> buffer;

  public Provider( int[] stream, Vector<Integer> buffer ) {

    this.stream = stream;
    this.buffer = buffer;
  }

  public void run() {

    for ( int i=0; i<stream.length; i=i+1 ) {

      int x = stream[i];
      synchronized ( buffer ) {
        while ( buffer.size() == 2 ) {
          try {
            buffer.wait();
          } catch ( Exception e ) {}
        }
        System.out.print( "P" );
        buffer.add(x);   // new element into buffer
        buffer.notify();
      }
    }
  }

}  // Provider
```

```
public class Task2 {

  public static void main( String[] args ) {

    int[] stream = new int[] { 1, 2, 3, 4, 3, 2 };

    // buffer initially empty
    Vector<Integer> buffer = new Vector<Integer>();
    Adder adder = new Adder( 3, buffer );
    Provider provider = new Provider( stream, buffer );

    adder.start();
    provider.start();
  }

}  // Task2
```

Problema 2

Considera le classi definite in Figura e immagina di avviare il programma attraverso il metodo main della classe Task2. Quale delle sequenze proposte sarà stampata al termine dell'esecuzione?

Opzioni possibili (una sola opzione selezionabile):

- P3P7P5
- P3PP7PP5P
- PP3P5P7
- PP3PP7PP5
- PPP375
- PPPPPP375
- Il programma si blocca in una situazione di stallo
- Il risultato non può essere previsto perché ci sono diverse possibilità

Problemi 3 e 4

Gli altri due problemi sono stati ispirati dal lavoro di Fekete[3]:

- il terzo problema ha lo scopo di vedere se gli studenti sono in grado di identificare, tra 5 opzioni, entrambi i frammenti di codice appropriati (equivalenti) per gestire una risorsa condivisa;
- il quarto problema richiede di riconoscere il codice soggetto a stallo (deadlock) e di impostare le correzioni adeguate.

[3] Ian D. Fekete. Teaching students to develop thread-safe java classes. SIGCSE Bull., 40(3):119–123, 6 2008.

Problema 3

Quali delle seguenti definizioni di metodi in Figura, all'interno di una classe che descrive l'implementazione di una risorsa condivisa, possono aiutare a evitare conflitti nella gestione della risorsa stessa? Sono selezionabili più opzioni.

Option 1

```
public synchronized int getValue() {
  return value;
}

public void setValue(int someValue) {
  value = someValue;
}

public void increment() {
  value++
}
```

Option 2

```
public int getValue() {
  return value;
}

public synchronized void
         setValue(int someValue) {
  value = someValue;
}

public void increment() {
  value++
}
```

Option 3

```
public synchronized int getValue() {
  return value;
}

public synchronized void
         setValue(int someValue) {
  value = someValue;
}

public synchronized void increment() {
  value++
}
```

Option 4

```
public int getValue() {
  return value;
}

public void setValue(int someValue) {
  value = someValue;
}

public void increment() {
  value++
}
```

Option 5

```
public int getValue() {
  synchronized (this) {
    return value;
  }
}

public void
  setValue(int someValue) {
  synchronized (this) {
    value = someValue;
  }
}

public void increment() {
  synchronized (this) {
    value++
  }
}
```

Figura: Problema 3

Problema 4

Considera un'istanza della classe *Bouncer* definita qui di seguito. Le modalità di sincronizzazione dei metodi *from1to2* e *from2to1* possono determinare situazioni di deadlock (stallo dell'esecuzione).

Quale tra i rimedi suggeriti sotto ritieni possano correggere il codice per evitare che si possa verificare uno stallo (pur garantendo una sincronizzazione appropriata)?

```
public class Bouncer {
    private Vector<Integer> seq1;
    private Vector<Integer> seq2;

    public Bouncer( Vector<Integer> seq1, Vector<Integer> seq2 ) {

        this.seq1 = seq1;
        this.seq2 = seq2;
    }

    public void from1to2() {
        synchronized ( seq1 ) {
            if ( seq1.size() == 0 ) {
                try {
                    seq1.wait();
                } catch ( Exception e ) {}
            }
            int item = seq1.elementAt(0);
            seq1.removeElementAt(0);
            synchronized ( seq2 ) {
                seq2.add( item );
                seq2.notify();
            }
        }
    }

    public void from2to1() {
        synchronized ( seq2 ) {
            if ( seq2.size() == 0 ) {
                try {
                    seq2.wait();
                } catch ( Exception e ) {}
            }
            int item = seq2.elementAt(0);
            seq2.removeElementAt(0);
            synchronized ( seq1 ) {
                seq1.add( item );
                seq1.notify();
            }
        }
    }
} // Bouncer
```

Problema 4

Opzioni possibili (una sola opzione selezionabile):

1. eliminare tutti i synchronized;
2. eliminare i synchronized annidati;
3. eliminare i synchronized da uno dei due metodi;
4. eliminare il synchronized esterno da uno dei metodi e quello annidato dall'altro;
5. trasformare i synchronized annidati in synchronized in sequenza (uno dopo l'altro anziché uno all'interno dell'altro);
6. invertire seq1 e seq2 in tutti i costrutti synchronized;
7. nessuna delle precedenti soluzioni.

Dati raccolti

Risultati

Il questionario è stato somministrato a **68** studenti del *quinto anno*, che hanno frequentano l'indirizzo Informatica in due istituti tecnici della nostra regione.

Gli studenti, a cui è stata insegnata la programmazione concorrente nell'anno scolastico precedente, sono stati impegnati nel compito in una situazione controllata, sotto la supervisione dei loro insegnanti.

Dovevano completare il questionario entro un'ora.

Risposte alle domande

Più di due terzi degli studenti trovano la programmazione di thread **difficile** (57%) o **molto difficile** (12%) e dichiarano di essere **insoddisfatti** (54%) o **completamente insoddisfatti** (13%) delle loro prestazioni in attività relative ai thread.

Comunque, la maggior parte di loro considera adeguati allo scopo sia il tempo impiegato per affrontare l'argomento (67%) sia i problemi di programmazione proposti (63%) dai loro insegnanti.

La **sincronizzazione** e la **gestione degli stati** di un thread sono considerate difficili da imparare rispettivamente dal 77% e dal 74% degli studenti, mentre diversi aspetti relativi all'organizzazione delle classi che implementano i thread sembrano meno critici.

Risultati dei problemi

Meno di un terzo degli studenti è più o meno fiducioso delle proprie risposte, il che conferma ancora una volta le loro difficoltà con la programmazione concorrente.

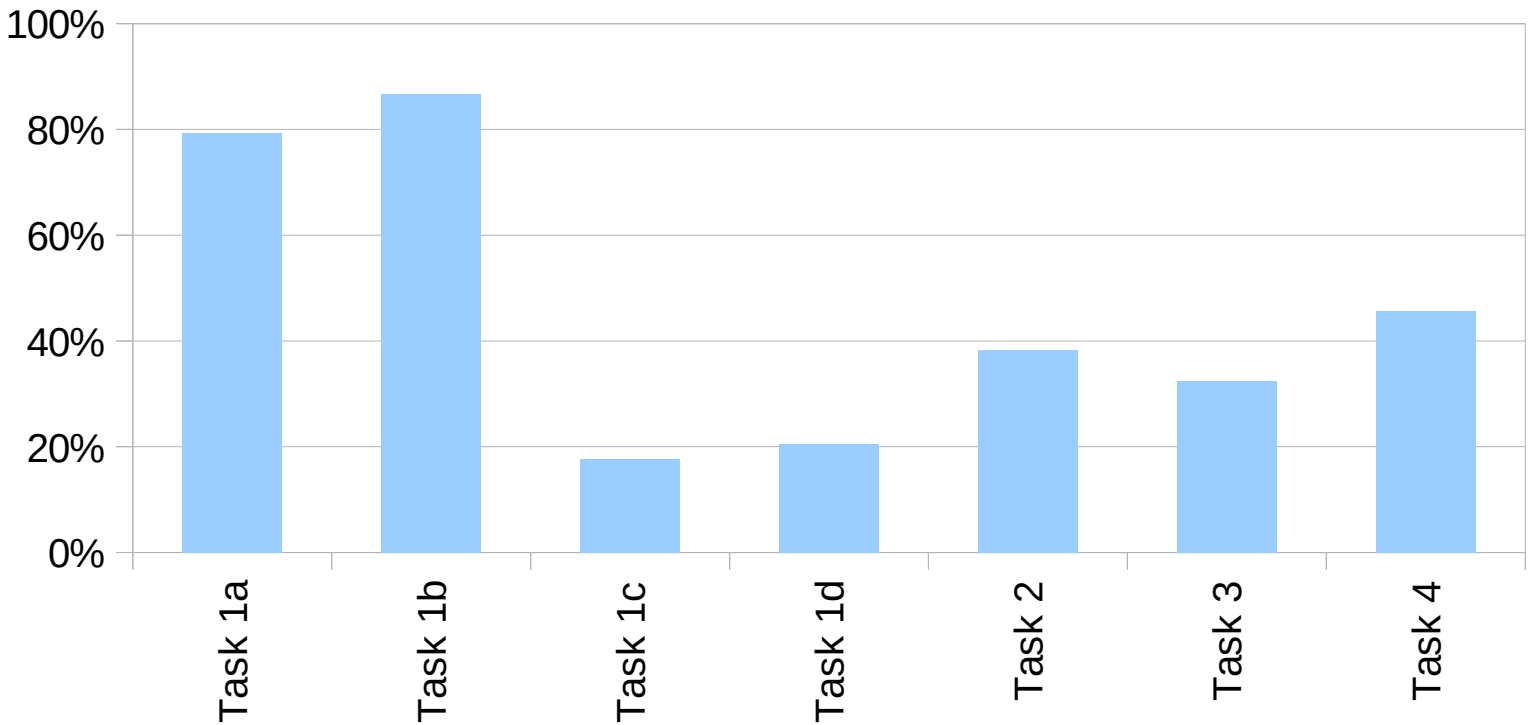

Figura: Le prestazioni degli studenti nei compiti proposti

Strumenti di aiuto

Le sezioni conclusive del sondaggio riguardano potenziali strumenti grafici/visivi che potrebbero aiutare a comprendere la programmazione concorrente, così come possibili suggerimenti aggiuntivi per migliorare la pratica didattica.

Il 59% degli studenti ha riferito di aver pensato all'utilizzo di diagrammi grafici e ben l'88% ritiene che una rappresentazione grafica potrebbe essere efficace per migliorare la loro comprensione della programmazione concorrente.

Diversi studenti non conoscono le reti di Petri (74%), i grafici Wait-for/Holt (51%), o gli automi a stati finiti (50%). Gli strumenti più votati sono stati i diagrammi a blocchi (44%) e i diagrammi di flusso (51%), anche se non sono i più adatti allo scopo.

Conclusioni

Conclusioni

La rilevanza del tema dal punto di vista professionale è indiscutibile, ciononostante gli studenti dimostrano difficoltà con un argomento che implica un alto livello di astrazione ed è difficilmente rappresentabile graficamente, ma di cui comunque il nostro sistema scolastico — a differenza di altri — prevede esplicitamente la trattazione.

Attualmente l'indagine delineata ha coinvolto 68 studenti, ma si prevede di allargare il campione. In particolare, al fine di estendere la portata di questa ricerca empirica, un **progetto ambizioso sarebbe riuscire a coinvolgere educatori che operano in contesti diversificati nel territorio nazionale**.

Grazie per l'attenzione