Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
Thread: esercizi

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

Esercizio 1

- Realizzare un sistema con un produttore e un consumatore
  - Usando i semafori, non i monitor
  - Tutta la sincronizzazione viene fatta nella coda
    - da realizzare usando come base quella vista a lezione

Esercizio 2

GIÀ VISTO

- Realizzare un sistema con un produttore e un consumatore
  - Tutta la sincronizzazione viene fatta nella coda (da realizzare)
  - Per la sincronizzazione si usano wait e notify.

Esercizio 3

- Realizzare un sistema con un produttore e un consumatore
  - Con BlockingQueue
    - con time out

Dalla documentazione

- https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/concurrent/BlockingQueue.html

offer

```
boolean offer(E e, long timeout, TimeUnit unit) throws InterruptedException
```

Inserts the specified element into this queue, waiting up to the specified wait time if necessary for space to become available.

Parameters:

`e` - the element to add

`timeout` - how long to wait before giving up, in units of `unit`

`unit` - a `TimeUnit` determining how to interpret the `timeout` parameter

Returns:

`true` if successful, or `false` if the specified waiting time elapses before space is available

Dalla documentazione

- https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/concurrent/BlockingQueue.html

poll

```
E poll(long timeout, TimeUnit unit) throws InterruptedException
```

Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available.

Parameters:

`timeout` - how long to wait before giving up, in units of `unit`

`unit` - a `TimeUnit` determining how to interpret the `timeout` parameter

Returns:

the head of this queue, or `null` if the specified waiting time elapses before an element is available

Throws:

`InterruptedException` - if interrupted while waiting

Esercizio 4

- Realizzare un sistema con due thread che condividono una risorsa
- I due thread devono accedere alternativamente alla risorsa.
  - Cioè non deve mai succedere che lo stesso thread acceda due volte consecutive alla risorsa.
- Protocollo:
  - Un thread richiede l'accesso alla risorsa quando gli serve
  - Usa la risorsa
  - Rilascia la risorsa

Esercizio 5 (tema d'esame del 22/6/2018)

- Si vuole simulare una partita di un gioco da tavolo.
- La partita coinvolge n giocatori (n>1).
- Ogni giocatore, finché la partita non è finita, quando è il suo turno tira i dadi e muove. Se in conseguenza della mossa vince la partita, termina. Se lanciando i dadi ha fatto doppio, tira nuovamente.
- Il codice dato implementa la logica descritta, ma è scorretto dal punto di vista della sincronizzazione.
- Si modifichi il codice dato in modo che il sistema si comporti correttamente.
- Turni: il giocatore 1 gioca per primo, poi gioca il giocatore 2, ecc.

Esercizio 6

- Realizzare produttore-consumatore
  - Usando BlockingQueue
  - Facendo polling:
    - Produttori e consumatori provano ad accedere
    - Se non è possibile riprovano dopo un po’