Sistemi Operativi – Prova del 21 Giugno 2011 - Tempo a disposizione 3 ore

Si deve progettare una struttura dati utile alla gestione delle merci presenti in un magazzino e accessibile da più thread in parallelo, i quali possono fare richieste per prelevare o depositare dei materiali dal magazzino stesso.

Strutture dati da progettare:

Magazzino: un Magazzino deve contenere un array *articoli[]*, composto da N elementi. Ogni elemento dell'array rappresenta la disponibilità in magazzino di un certo articolo (esempio, *articolo[3]* = 10, indica che in magazzino sono presenti 10 pezzi dell'articolo con codice 3). Ogni articolo è sistemato su uno scaffale che non consente di depositare più di 100 pezzi per articolo.

ListaMateriali: una lista materiali è un'elenco di coppie <CodArticolo,Quantità>, dove CodArticolo rappresenta il codice di un certo articolo e Quantità la quantità corrispondente.
Le Liste di Materiali rappresentano ordini di materiali che si vogliono prelevare (in uscita dal magazzino), oppure ordini di materiali che sono in ingresso al magazzino (approvvigionamenti).

ESEMPIO: La lista materiali <1,5>,<2,7>,<3,10> può essere utilizzata per rappresentare un ordine corrispondente a una richiesta di 5 pezzi per l'articolo 1, 7 pezzi per l'articolo 2, e 10 pezzi per l'articolo 3.

La struttura dati Magazzino deve essere accessibile attraverso i metodi pubblici:

ListaMateriali* putMateriali(ListaMateriali* L)
Deposita nel magazzino tutti i materiali presenti nella lista L. La ListaMateriali ritornata deve elencare tutti quegli articoli (e relative quantità) in eccedenza rispetto alla quantità massima di 100 pezzi che può essere depositata. La lista ritornata è vuota se è stato possibile depositare tutti i materiali elencati in L.

ESEMPIO: supponiamo che il magazzino contenga 89 pezzi dell'articolo 1, 3 pezzi dell'articolo 2, e 50 pezzi dell'articolo 3; supponiamo di invocare **putMateriali(H)**, dove H è una ListaMateriali che elenca 20 pezzi per l'articolo 1, 20 pezzi per l'articolo 2 e 20 pezzi per l'articolo 3. All'uscita da putMateriali il magazzino conterrà le seguenti merci:

Articolo 1 : 100 unità
Articolo 2 : 23 unità
Articolo 3 : 70 unità

putMateriali restituisce una lista di un solo elemento, rappresentata dalla coppia <1,9>, indicante che sono rimasti 9 pezzi in eccedenza per l'articolo 1.

ListaMateriali* getMateriali(ListaMateriali* L)

getMateriali riceve come parametro un ordine L (cioè una ListaMateriali), prelevando i materiali corrispondenti dal magazzino. Nel caso in cui un certo articolo non sia disponibile nella quantità richiesta, sono previsti due casi:

1. Se l'articolo richiesto risulta totalmente esaurito, il thread chiamante viene bloccato e posto in attesa che sia disponibile almeno una unità dell'elemento;
2. Se l'articolo richiesto risulta parzialmente disponibile, vengono fornite tutte le unità disponibili, senza bloccare il thread chiamante (è possibile dunque che l'ordine non venga esaudito completamente e getMateriali restituisca una lista di materiali incompleta).

getMateriali deve ritornare la lista degli articoli e quantità effettivamente prelevate dal magazzino.

ESEMPIO: se il magazzino si trova nel seguente stato:

Articolo 1 : 100 unità
Articolo 2 : 23 unità
Articolo 3 : 70 unità

E viene effettuata una richiesta di 80 unità per l'articolo 3 e 10 unità per l'articolo 1, allora **getMateriali** ritorna la lista <1,10>,<3,70> senza bloccarsi.

Se invece il magazzino è nello stato:

Articolo 1 : 100 unità
Articolo 2 : 23 unità
Articolo 3 : 0 unità

Lo stesso ordine visto sopra (<1,10>,<3,80>) provoca *la messa in attesa del thread chiamante di **getMateriali** fintantochè la disponibilità dell'articolo 3 non diventi di almeno una unità*.

Si implementi inoltre un thread **Visualizzatore** che mostri periodicamente a video lo stato del magazzino, e, a scelta dello studente, dei thread che simulino l'accesso periodico e casuale al magazzino invocando **getMateriali** e **putMateriali.**

getMateriali e **putMateriali** devono essere implementate garantendo (in stretto ordine di priorità)

1. Mutua esclusione nell'accesso ai dati condivisi;
2. Assenza di situazioni di deadlock;
3. Assenza di situazioni di starvation;
4. Massimo parallelismo (ad es. bloccare due getMateriali invocate in parallelo solo quando accedono allo stesso articolo).

*E' **parte integrante** della prova di esame completare le specifiche date nei punti non esplicitamente definiti, introducendo tutte le strutture dati che si ritengano necessarie, e risolvendo eventuali ambiguità. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.*