**Corso di Sistemi Operativi – Appello straordinario del 21 Gennaio 2012**

Si deve progettare una tipologia di buffer FIFO che chiameremo CodaCombinata. Tale struttura dati deve essere accessibile da più thread contemporaneamente, e deve disporre di metodi di inserimento e prelievo bloccanti. Per “bloccante” si intende che l’operazione di “get” pone il thread chiamante in attesa se non sono disponibili elementi, e l’operazione di “put” pone il thread chiamante in attesa se non ci sono posti disponibili nel buffer. Una CodaCombinata è composta da **due buffer FIFO** separati A0 e A1, entrambi di dimensione N. Ogni buffer può contenere elementi del tipo generico T. In ogni momento uno dei due buffer riveste il ruolo di buffer Primario e l’altro riveste il ruolo di buffer Secondario. Inizialmente Primario=A0, e Secondario=A1. I metodi che una CodaCombinata deve mettere a disposizione sono:

-scambiaPrimario() : inverte i ruoli dei due buffer, ad esempio se in un certo momento Primario=A0 e Secondario=A1, allora la configurazione deve diventare Primario=A1, Secondario=A0.

-put(T A) : inserisce l’elemento A nel buffer SECONDARIO. Se quest’ultimo dovesse essere pieno, si estrae un elemento H dal buffer SECONDARIO, lo si inserisce nel buffer PRIMARIO, e si sfrutta il posto appena liberato nel buffer SECONDARIO per inserire A. Se anche il buffer PRIMARIO dovesse essere pieno, si pone il thread chiamante in attesa che si liberi un posto in uno dei due buffer, per poi procedere come specificato.

-T get(): estrae un elemento dal buffer PRIMARIO: se questo dovesse risultare vuoto, l’elemento viene estratto dal buffer SECONDARIO. Se entrambi i buffer sono vuoti, il thread chiamante deve essere posto in attesa fino a che non sia presente qualche elemento in uno dei due buffer.

-putPrimario(A): inserisce un elemento nel buffer primario: blocca il thread chiamante finchè il buffer Primario resta pieno.

-putSecondario(A): inserisce un elemento nel buffer secondario: blocca il thread chiamante finchè il buffer Secondario resta pieno.

-T getPrimario(): estrae un elemento dal buffer primario: blocca il thread chiamante finchè il buffer Primario resta vuoto.

-T getSecondario(): estrae un elemento dal buffer secondario: blocca il thread chiamante finchè il buffer secondario resta vuoto.

La libreria deve essere implementata garantendo **la mutua esclusione** nell’accesso ai dati condivisi **solo *ove necessario****,* al fine di garantire il massimo grado di parallelismo;

*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.*

***La prova può essere svolta in qualsiasi linguaggio di programmazione a scelta*** *dotato di costrutti di supporto alla programmazione multi-threaded (esempio, C++ con libreria JTC, Java). Nel caso dovessero servire al candidato, si può assumere di avere a disposizione i costrutti di sincronizzazione lock, read/write lock, barriera e buffer sincronizzato. Non è pertanto necessario fornire l’implementazione di tali classi, ma solo eventuali loro specializzazioni ottenute per ereditarietà.*

*Non è esplicitamente richiesto di scrivere un main() o di implementare esplicitamente dei thread che utilizzino una CodaCombinata.*

**Suggerimenti del docente (aggiunti dopo l’esame)**

Si noti che sia A0, A1 e la CodaCombinata sono gestiti singolarmente con politica FIFO (come è detto esplicitamente nella traccia).

Come prima cosa conviene progettare (o riutilizzare il codice visto durante il corso, o utilizzare le strutture dati standard Java) una classe Buffer non sincronizzata e dotata dei normali metodi put e get, in versione *NON BLOCCANTE:* anziché porre in wait il thread chiamante, put e get ritornano una indicazione (e.g. attraverso un valore booleano) che riporta se l’operazione è avvenuta correttamente, o se il buffer risulta rispettivamente pieno (non è possibile put) o vuoto (non è possibile get).

scambiaPrimario() è realizzabile prevedendo in CodaCombinata i 4 campi A0, A1, Primario e Secondario.

Si pone inizialmente Primario=A0 (si suppone che A0,A1, Primario e Secondario siano riferimenti Java, oppure puntatori C++) e Secondario=A1. Per implementare scambiaPrimario è sufficiente scambiare le variabili Primario e Secondario. Il metodo deve essere ovviamente sincronizzato.

**Errori più comuni:**

* Effettuare lo scambio tra primario e secondario copiando il contenuto di A0 su A1 e viceversa. Una soluzione estremamente inefficiente, ma che soprattutto in molti casi è stata implementata anche male: **non basta scambiare il contenuto di A0 e A1, ma è necessario scambiare anche tutte le variabili di servizio associate ad A0 e A1 (in, out, ecc. ecc.)**
* Dimenticarsi qualsiasi signal() o notify() che svegli opportunamente i thread in attesa, oppure fare signal() su una condition mentre i thread aspettano su un’altra.
* Fare unlock() prima di porsi in wait su una condition. Non solo non serve, poiché il lock viene rilasciato automaticamente quando si invoca await(), ma è SCORRETTO poiché, *1) non si può invocare await() se non si è in possesso del lock corrispondente, e 2) se si lascia il lock, si da ad altri thread la possibilità di acquisirlo all’interno di un pezzo di codice che ci si aspetta sia eseguito in mutua esclusione.*
* *Non riverificare la condizione di attesa in uscita da wait(): while(devoAspettare) wait() 🡺 OK, if(devoAspettare) wait() 🡺 ERRATO QUASI SEMPRE.*
* *Non riusare il codice: esempio, getPrimario e getSecondario dovrebbero essere lunghe al massimo un paio di righe (invocando get() sul buffer opportuno), e NON RE-IMPLEMENTARE due volte il meccanismo di get() da un buffer.*
* *Piccole cose: if (primario == true) primario = false else primario = true; 🡺 MEGLIO: primario = !primario.*