QUIZ

Nei modelli della concorrenza con consistenza sequenziale
1. Bisogna considerare sono le computazioni con scheduling dei thread basato su strategie preemptive
2. Tutti i thread vengono eseguiti almeno una volta
3. Viene garantita l'assenza di deadlock e l'assenza di starvation
4. Thread differenti non possono condividere il proprio stack

Quando si utilizza un semaforo in un programma concorrente
1. i thread possono ancora accedere simultaneamente alla propria sezione critica
2. l'accesso a dati condivisi da parte di più thread è sempre serializzato
3. un thread che esegue l'operazione down incrementa il contatore interno al semaforo di uno
4. è opportuno inizializzare il semaforo a 0 se si vuole puoi usare il semaforo come mutex

Quando si utilizza un thread pool
1. i thread eseguono task prelevandoli da uno stack
2. alla fine dell'esecuzione di un task il corrispondente thread rimane attivo
3. non bisogna preoccuparsi della gestione della garbage collection della memoria
4. non bisogna preoccuparsi della creazione dei thread

Un Semaforo Generale
1. è un semaforo il cui contatore interno può assumere un valore maggiore o uguale a zero
2. è implementato come un oggetto immutabile in Java
3. non può essere usato come campo di una concurrent HashMap
4. contiene al suo interno una coda di thread

La tecnica di programmazione basata su confinamento per thread
1. viene usata per ridurre l'uso di lock nei programmi concorrenti
2. viene usata per rendere efficiente la gestione della memoria nei programmi concorrenti
3. viene usata nell'implementazione di server multithreaded
4. viene usata nell'implementazione dei monitor alla Hoare

Una barriera di memoria o memory fence
1. risolve il problema della sezione critica
2. Viene sempre invocata alla fine di blocchi sincronizzati in Java
3. può essere usata per garantire mutua-esclusione in architetture debolmente consistenti
4. ha come effetto quello di disabilitare per più cicli di esecuzione tutte le interruzioni hardware

Il problema della sezione critica
1. si applica a programmi concorrenti qualsiasi
2. richiede di soddisfare almeno le proprietà di mutua esclusione e assenza di starvation
3. non richiede particolari assunzioni sulla struttura della sezione critica
4. è formulato per programmi concorrenti con un numero arbitrario ma finito di thread

In un programma concorrente con un input fissato
1. due diverse computazioni possono eseguire infinite volte la stessa istruzione
2. se una computazione termina allora tutte le possibili computazioni terminano
3. se una computazione non termina allora tutte le possibili computazioni non terminano
4. due computazioni diverse possono dare lo stesso risultato

Nell'esecuzione di un programma concorrente
1. tutti i thread lanciati da un programma partono sempre simultaneamente
2. i thread vengono eseguiti sempre in parallelo
3. non è possibile alternare context-switch di due diversi thread
4. il numero di context-switch non dipende dallo scheduler

Quando usiamo oggetti runnable in Java
1. Le chiamate dei metodi corrispondenti possono restituire valori
2. Le chiamate dei metodi corrispondenti sono effettuate in mutua esclusione
3. Le chiamate dei metodi corrispondenti sono tutte effettuate in maniera asincrona
4. Non possiamo propagare le eccezioni al di fuori dei metodi corrispondenti

La libreria SynchronizeCollection di Java
1. Viene usata per incapsulare strutture dati per renderle threadsafe
2. Viene usata come alternativa agli ExecutorService
3. Ha un metodo “wait” che viene usato per sincronizzare thread
4. Ha un metodo “notify” che viene usato per sbloccare thread in attesa

Un processo nei sistemi operativi:
1. può condividere codice con altri processi
2. è un’unità di allocazione sia di risorse che di esecuzione
3. ha sempre accesso allo spazio di indirizzamento kernel
4. gestisce al più un program counter

Si verifica sempre un context switch quando
1. un processo ritorna da una chiamata di procedura
2. un processo ritorna da una chiamata di funzione
3. un processo esegue un metodo di un’oggetto condiviso
4. un processo esegue una routine di gestione di un’interrupt

Un variabile condition
1. si può usare solo dopo aver acquisto un lock
2. definisce una condizione che viene controllata in maniera atomica
3. mette in attesa il thread che invoca la corrispondente operazione signal
4. mette in attesa il thread che invoca la corrispondente operazione signal se ci sono altri thread in attesa

Il Thread Pool
1. è una struttura dati sincronizzata
2. è una struttura dati fornita a livello kernel
3. è stato introdotto per creare thread già forniti di lock di sincronizzazione
4. è stato introdotto per distinguere task e thread

Nelle librerie per UI come Swing
1. i singoli thread possono modificare campi delle strutture dati della UI
2. un gestore di UI (EDT) è un server multithreaded
3. ogni widget di una GUI è gestito da un diverso EDT
4. per default viene garantita la mutua esclusione nella modifica della UI

Una Barriera di Memoria (Memory Fence)
1. forza il completamento di operazioni di scrittura in attesa
2. forza la terminazione del thread che la esegue
3. è equivalente ad un lock
4. assicura l’assenza di starvation

Un thread
1. è una unità di allocazione di esecuzione
2. non può condividere la memoria con altri thread
3. non può essere usato per eseguire una parte di un calcolo parallelo
4. permette l’esecuzione asincrona di funzioni e procedure

Un context switch si può verificare
1. quando un processo ritorna da una chiamata di procedura
2. quando un processo crea un nuovo thread
3. quando un processo termina un’altro processo
4. quando un processo esegue codice user

Una Condition Variable
1. va usata solo all’interno di operazioni del monitor in cui è definita
2. è una condizione che viene valutata in maniera asincrona rispetto al programma chiamante
3. gestisce code interne ad un monitor
4. garantisce write atomicity

Una Barriera di Memoria (Memory Fence)
1. forza la terminazione di tutti i thread in esecuzione tranne il main
2. forza la terminazione del main ma non di altri thread in esecuzione
3. controlla eventuali ottimizzazioni nella gestione della memoria
4. assicura l'assenza di starvation

Il Thread Pool
1. è una struttura dati sincronizzata
2. è stato introdotto nella programmazione concorrente per rendere più efficiente un programma
3. è stato introdotto nella programmazione concorrente per evitare possibili deadlock
4. è stato introdotto nella programmazione concorrente per eseguire blocchi di codice sincrono

Un Reentrant Lock
1. è stato introdotto nella programmazione concorrente come semaforo per oggetti di classe
2. viene usato per garantire la mutua esclusione tra thread con variabili condivise
3. viene usato per evitare deadlock in una chiamata ricorsiva di un metodo thread-safe
4. garantisce sempre starvation-freedom se usato per controllare una risorsa condivisa

Nei modelli della concorrenza con consistenza forte (strong consistency):
1. Vale sempre la single-thread rule
2. È possibile che istruzioni all'interno dello stesso thread vengano eseguite out-of-order
3. Non viene garantita la write-atomicity nelle operazioni di update
4. Thread differenti non possono modificare le stesse aree di memoria

L'assenza di race-condition
1. si ottiene dichiarando thread che non allocano dati sullo heap
2. si ottiene creando thread che allocano memoria solo attraverso variabili locali
3. si ottiene garantendo che i dati siano letti o modificati solo all'interno di singoli thread
4. si ottiene acquisendo lock su dati condivisi tra diversi thread

Quando si utilizza un monitor alla Hoare in Java
1. i thread non condividono memoria ma vengono usati solo per parallelizzare il calcolo
2. i context-switch avvengono solo all'uscita dal monitor
3. se un thread sospende l'esecuzione con "wait" entrerà nel monitor alla prima chiamata di "notify"
4. un thread prima di chiamare "wait" rilascia il lock sul monitor

Un ReadWrite lock
1. è un semaforo binario che viene usato su oggetti con metodi getter e setter
2. viene usato per garantire la mutua esclusione tra thread che aggiornano una variabile condivisa
3. viene usato per garantire la mutua esclusione tra thread che leggono una variabile condivisa
4. garantisce starvation-freedom se usato per controllare una risorsa condivisa

Un Array CopyOnWrite in Java
1. viene usato per ottimizzare le operazioni di update di celle di un array
2. implementa un'istanza del produttore-consumatore
3. fornisce solo operazioni thread-safe per modificare un array concorrente
4. implementa la tecnica dello snapshot per strutture dati concorrenti

La tecnica di programmazione chiamata lock-splitting
1. viene usata per includere in un singolo record diversi semafori binari
2. viene usata per massimizzare la concorrenza mantenendo consistenza dei dati
3. viene usata per minimizzare il numero di lock in un programma concorrente
4. viene usata nell'implementazione di liste concorrenti

Le soluzioni al problema della sezione critica viste a lezione
1. possono essere composte da thread che non usano mai la risorsa condivisa da proteggere
2. si possono implementare senza usare lock o altre istruzioni atomiche specifiche dell'hw
3. sono sempre corrette indipendentemente dall'architettura hw sottostante
4. dipendono dalla velocità nell'esecuzione dei diversi thread

In un programma concorrente
1. un errore su un certo input si può riprodurre ripetendo l'esecuzione una sola volta
2. la funzione calcolata dal programma associa ad un certo input un insieme di output
3. con lo stesso input posso avere un'esecuzione che termina e una che non termina
4. con lo stesso input posso avere un'esecuzione che termina e una che si blocca

Nell'esecuzione di un programma concorrente
1. i thread vengono sempre eseguiti su core diversi
2. i thread vengono sempre eseguiti in parallelo
3. non avvengono mai context-switch sullo stesso core
4. il numero di context-switch dipende dal numero di lock usati nel programma

Consideriamo un programma concorrente con due thread T1 e T2
1. servono almeno due lock per generare un deadlock nell'esecuzione di T1 e T2
2. con un solo lock è possibile fare in modo che T1 venga eseguito tutto insieme prima o dopo T2
3. servono almeno due lock per fare in modo che T1 venga sempre eseguito prima di T2
4. il numero di possibili esecuzioni dipende solo dal numero di istruzioni che operano su dati condivisi

Nella libreria RMI
1. L'accesso ad un oggetto remoto è sempre thread-safe
2. Il registry viene gestito dallo stesso server che gestisce un oggetto remoto
3. Il registry serializza le chiamate dei metodi verso un oggetto remoto
4. L'implementazione di un'interfaccia remota deve essere la stessa su server e client

Nei modelli della concorrenza con consistenza sequenziale:
1. vengono ammesse solo computazioni terminanti
2. vengono ammesse esecuzioni concorrenti che non rispettano program order
3. viene sempre garantita l'assenza di deadlock
4. vengono ammesse esecuzioni concorrenti con interleaving di eventi di diversi thread

La proprietà di write atomicity
1. garantisce l'assenza di race condition
2. garantisce che le operazioni di scrittura su una struttura dati siano eseguite in maniera atomica
3. assicura che tutte le operazioni di scritture siano viste nello stesso ordine da tutti i thread
4. è sempre garantita in architetture debolmente consistenti

Quando si utilizza una struttura dati concorrente
1. per ogni accesso alla struttura viene garantita la mutua esclusione
2. per ogni sequenza di accessi alla struttura dati viene garantita la mutua esclusione
3. non bisogna preoccuparsi della de-allocazione della struttura dati
4. l'implementazione delle operazioni utilizza sempre un lock globale

Una ConcurrentHashMap in Java
1. non può essere usata in blocchi sincronizzati
2. fornisce metodi thread-safe per inserimento e ricerca
3. se usata in un programma concorrente garantisce l'assenza di race condition sui propri dati
4. è un'implementazione della tecnica di confinamento per thread

Un reentrant lock
1. è un semaforo usato per oggetti con metodi che restituiscono valori
2. viene usato per garantire la mutua esclusione tra thread con variabili condivise
3. viene usato per evitare deadlock in una chiamata ricorsiva
4. garantisce starvation-freedom se usato per controllare una risorsa condivisa

Una barriera di memoria o memory fence
1. risolve il problema della sezione critica
2. può essere invocata alla fine di blocchi sincronizzati
3. può essere usata per garantire mutua-esclusione in architetture debolmente consistenti
4. ha come effetto quello di disabilitare le interruzioni hardware

Il problema della sezione critica
1. si può applicare a programmi dove la sezione critica può contenere un programma qualsiasi
2. richiede di soddisfare solo le proprietà di mutua esclusione e assenza di starvation
3. è formulato per thread eseguiti tutti con la stessa velocità
4. è formulato per programmi concorrenti con al più due thread di esecuzione

Quando usiamo oggetti runnable in Java
1. Le chiamate dei metodi corrispondenti possono restituire valori
2. Le chiamate dei metodi corrispondenti sono effettuate in mutua esclusione
3. Le chiamate dei metodi corrispondenti sono tutte effettuate in maniera asincrona
4. Non possiamo propagare le eccezioni al di fuori dei metodi corrispondenti

Nella libreria RMI
1. L'accesso ad un oggetto remoto è sempre thread-safe
2. Il registry viene gestito dallo stesso server che gestisce un oggetto remoto
3. Il registry serializza le chiamate dei metodi verso un oggetto remoto
4. L'implementazione di un'interfaccia remota deve essere la stessa su server e client

Quando si utilizza una struttura dati concorrente
1. per ogni accesso alla struttura viene garantita l'assenza di race condition
2. per ogni sequenza di accessi alla struttura dati viene garantita l'assenza di race condition
3. per ogni programma viene garantita l'assenza di race condition
4. per ogni operazione su un suo elemento (es un oggetto in una lista) viene garantita l'assenza di race condition

Nei modelli della concorrenza con interleaving
1. vengono ammesse solo computazioni che rispettano program order
2. vengono ammesse solo computazioni ottenute eseguendo in blocco tutte le istruzioni di ogni singolo programma in un ordine casuale (es programma 2, programma 1, programma 3)
3. viene sempre garantita la mutua esclusione
4. il numero di context switch è limitato superiormente da una costante

Una Synchronized Collection in Java
1. è equivalente ad una barriera di memoria
2. rende sincronizzata una struttura dati non concorrente
3. implementa il pattern singleton in versione thread-safe
4. è un'implementazione della tecnica di confinamento per oggetto

Un mutex
1. è un semaforo con due possibili valori ed uno stack di thread in attesa
2. è un semaforo con tre possibili valori ed una coda di thread in attesa
3. viene usato per evitare riordinamenti di istruzioni in un thread
4. garantisce starvation-freedom se usato per controllare una risorsa condivisa

Un'operazione atomica
1. è sempre implementata tramite lock
2. non è compatibile con il modello di concorrenza basato su consistenza sequenziale
3. elimina possibili interferenze tra diversi thread fino alla fine della sua esecuzione
4. può utilizzare istruzioni hardware per garantire assenza di race condition

Una barriera di sincronizzazione
1. risolve il problema della sezione critica
2. può essere invocata alla fine di blocchi sincronizzati
3. definisce un punto di sincronizzazione all'interno di codice di singoli thread
4. ha come effetto quello di disabilitare le interruzioni hardware

Il problema della sezione critica
1. si applica a programmi che non condividono memoria
2. si risolve scegliendo in maniera casuale un thread (tra quelli in attesa)
3. si risolve applicando Peterson a due thread scelti casualmente (tra quelli in attesa)
4. è formulato per programmi concorrenti con due o più thread di esecuzione

Quando usiamo oggetti callable in Java
1. Le chiamate dei metodi corrispondenti possono restituire valori
2. Le chiamate dei metodi corrispondenti sono effettuate in mutua esclusione
3. Le chiamate dei metodi corrispondenti sono tutte effettuate in maniera sincrona
4. Non possiamo propagare le eccezioni al di fuori dei metodi corrispondenti

Nella libreria RMI
1. Ogni programma può esportare un solo oggetto remoto
2. La comunicazione è sempre unidirezionale
3. I parametri di un metodo remoto non devono essere necessariamente oggetti serializzabili
4. L'implementazione dell'interfaccia di un parametro deve essere la stessa su server e client

Un processo nei sistemi operativi
1. è un'unità di allocazione sia di risorse che di esecuzione
2. può condividere lo stack con altri processi
3. può essere usato per eseguire una parte di un calcolo parallelo
4. può essere eseguito in modalità user e modalità kernel

Un context switch si può verificare
1. quando un processo ritorna da una chiamata di sistema
2. quando un processo ritorna da una chiamata di funzione
3. quando un processo si sospende su un'operazione di I/O
4. Quando un processo esegue una routine di gestione di interrupt

Un variabile condition
1. si può usare in qualsiasi parte del codice
2. ha un'operazione di signal che è equivalente alla up dei semafori
3. è un flag di tipo Bool dichiarato volatile
4. garantisce write atomicity nei programmi dove viene usata

Nel debugger di Eclipse si usano i trigger condizionali
1. per controllare condizioni su uno specifico thread
2. per forzare la generazione di context switch
3. per attivare routine di interrupt
4. per stampare messaggi di profiling utili allo sviluppatore

Un Executor
1. è un'implementazione di un monitor fornita da Java
2. viene usato per garantire la mutua esclusione tra thread con variabili condivise
3. viene usato per evitare deadlock
4. viene usato come costruttore di thread pool

Una BlockingQueue in Java
1. ha metodi thread safe
2. è una struttura dati sincronizzata
3. può essere usata per implementare un thread pool
4. non può essere usata come campo di una struttura dati sincronizzata

Lo schema di codice in C
if (fork() != 0) { … } else { execve(command, parameters, NULL); }
1. viene usato per creare un thread Unix/Linux
2. si può usare per assegnare un task ad un thread in Unix/Linux
3. si può usare per eseguire in parallelo due parti di codice
4. si può usare come blocco base per usare in maniera efficiente un'architetture multicore

Un reentrant lock
1. viene usato per garantire la mutua esclusione tra thread con variabili condivise
2. garantisce sempre assenza di deadlock
3. è un lock esterno con memory fence
4. è interscambiabile con semafori binari

Una variable condition
1. non mette mai in attesa il thread che invoca la corrispondente operazione signal
2. definisce una condizione che potrebbe non essere controllata in maniera atomica
3. mette in attesa il thread che invoca la corrispondente operazione signal solo quando esiste un thread in attesa
4. si può usare solo dopo aver acquisito un lock

Il Thread Pool
1. è una struttura dati fornita a livello kernel
2. esegue tutti i task in maniera sequenziale se essi condividono almeno un oggetto
3. è una struttura dati sincronizzata
4. permette di schedulare task concorrenti controllando i thread sottostanti

La tecnica di programmazione concorrente chiamata lock-splitting
1. consiste nell'assegnare ad ogni thread una unica operazione su un lock consiviso (es acquire a T1 e release a T2)
2. è simile all'uso di barriere di sincronizzazione
3. può essere usata solo nell'implementazione di lsite
4. viene usata per massimizzare la concorrenza mantenendo consistenza dei dati

Una barriera di memoria (memory fence)
1. non è altro che un lock per acquisire una risorsa condivisa
2. forza il completamento di operazioni di scrittura in attesa
3. garantisce sempre assenza di deadlock
4. ha come uso principale la definizione di un meating point per un insieme di thread

Si verifica sempre un context switch quando:
1. un processo esegue metodo sincronizzato di un'oggetto condiviso
2. un progetto esegue una chiamata di sistema bloccante
3. un processo ritorna da una chiamata di procedura
4. un processo ritorna da una chiamata di funzione

Nella libreria RMI:
1. il registry serializza le chiamate dei metodi verso un oggetto remoto
2. l'implmentazione di un interfaccia remota non deve essere la stessa sul server
3. l'accesso ad un oggetto remoto e sempre thread safe
4. il registry viene gestito dallo stesso server che gestisce un oggetto remoto