PCAD a.a. 2018/19 - Scritto del 14 giugno 2019

L'esame è composto da domande a risposta multipla ed esercizi a risposta libera.
Se richiesto dal testo o se avete dubbi sulla formulazione di una domanda aggiungete una breve spiegazione per giustificare la risposta. Nella stessa domanda ci può essere più di un'affermazione vera. Occorre raggiungere almeno 15 punti per poter far media con il voto della discussione del progetto.

D1 (1 punto) 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

D2 (1 punto) 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

D3 (1 punto) 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

D4 (1 punto) 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

D5 (1 punto) 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

D6 (1 punto) 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

D7 (2 punti) 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

D8 (2 punti) 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

D9 (2 punti) 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

D10 (4 punti) Considerare il seguente codice C:

```
int idx=0;
void *foo(void *vargp) {
  idx++;
  printf("Thread %d\n", idx);
  return(0);
}

int main() {
  pthread_t tid[5];
  void *ret;
  for (int i = 0; i < 5; i++) {
     pthread_create(&tid[i], 0, foo, 0);
  }

  for (int i = 0; i < 5; i++) {
     pthread_join(tid[i],&ret);
  }
  return 0;
}
```

1. Il programma può produrre l'output 1 2 3 4 5? (motivare la risposta)

2. Il programma può produrre l'output 0 1 2 3 4? (motivare la risposta)

3. Il programma può produrre l'output 2 2 5 5 5? (motivare la risposta)

4. Il programma NON può produrre l'output 5 5 5 5 5? (motivare la risposta)

Esercizio 1 (8 punti)
Descrivere una possibile implementazione in Java di un Thread Pool senza usare Executors etc.

Esercizio 2 (8 punti)
Descrivere in pseudo-codice una possibile implementazione dell'algoritmo distribuito di Mattern/Fidge per la sincronizzazione di clock logici vettoriali.