PCAD a.a. 2018/19 - Scritto 11 settembre 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) 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

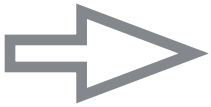
D2 (1 punto) Un context switch si può verificare
1. quando un processo ritorna da una chiamata di procedura
2. quando un processo crea un nuovo thread
3. quando un processo termina un'altro processo
4. quando un processo esegue codice user

D3 (1 punto) 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

D4 (1 punto) 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

D5 (1 punto) 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

D6 (1 punto) 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



D7 (2 punti) Confrontare (con brevi spiegazioni) il comportamento dei seguenti programmi C

Programma A

```
int idx=0;

void *foo(void *aux) {
 idx=12;
 return(0);
}

int main() {
  pthread_t tid;
  void *ret;
  pthread_create(&tid, 0, foo, 0);
  idx=8;
  pthread_join(tid,&ret);
  printf("Thread %d\n", idx);
  return 0;
}
```

Programma B

```
int idx=0;

void *foo(void *aux) {
 idx=12;
 return(0);
}

int main() {
  pthread_t tid;
  void *ret;
  pthread_create(&tid, 0, foo, 0);
  pthread_join(tid,&ret);
  idx=8;
  printf("Thread %d\n", idx);
  return 0;
}
```

Risposta:

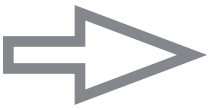
D8 (2 punti) Considerate il programma concorrente in pseudo codice

Shared Memory: semaphor s1=s2=1;
Thread T1: { down(s1); down(s2); print("a"); }
Thread T2: { down(s2); down(s1); print("b"); }

1. Quando il programma viene eseguito stampa sempre sia "a" che "b"
2. " " " stampa solo una delle due lettere tra "a" e "b"
3. " " " non va mai in deadlock
4. Il programma è thread safe

D9 (2 punti) Nella libreria RMI una callback

1. è sempre thread-safe
2. è una chiamata di un particolare metodo di un client eseguita ad ogni connessione con il server
3. è una chiamata di un particolare metodo del server eseguita ad ogni richiesta di un client
4. è una chiamata di un particolare metodo di un client eseguita ad ogni errore di connessione in rete



D10 (4 punti) Considerare il seguente schema di codice ispirato alla sintassi di Java:

```
class mydata{
   private a=0;
   public synchronized m1(){ wait();  a=1; }
   public synchronized m2(){ if (a = = 0) notify();  }
   public synchronized m3(){ print(a);  }
}
```

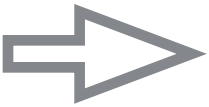
Thread 1 (con reference ad A): A.m1(); A. m2(); A.m3();
Thread 2 (con reference ad A e B): B.m2(); A.m3(); B.m3();
Main:
mydata A = new mydata();
mydata B = new mydata();
creazione con reference ad A ed avvio di un'istanza di Thread 1;
creazione con reference ad A e B ed avvio di un'istanza di Thread 2;

1. Il programma può stampare 0? (motivare la risposta)

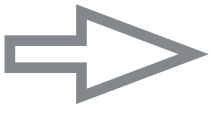
2. Il programma può andare in deadlock? (motivare la risposta)

3. Il programma può andare in deadlock se Thread 2 non viene mai eseguito? (motivare la risposta)

4. Il programma può stampare 1? (motivare la risposta)



Esercizio 1 (8 punti)
Scrivere una possibile soluzione (cioè un programma concorrente nel linguaggio che preferite) al problema del produttore e consumatore con buffer potenzialmente unbounded.



Esercizio 2 (8 punti)
Descrivere un esempio di gestione di una struttura dati distribuita tra diversi nodi di una rete prendendo spunto da esempi di algoritmi distribuiti o esercizi di laboratorio visti nel corso.