PCAD a.a. 2018/19 - Scritto 3 febbraio 2020

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 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

V V F F

D2 (1 punto) 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

F F F V

D3 (1 punto) 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

V F F F

D4 (1 punto) 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

F F F V

D5 (1 punto) 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

V F F F

D6 (1 punto) 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

V F F F

D8 (4 punti) Considerate il programma concorrente in pseudo codice

Shared Memory: semaphore s1=2; s2=1;
Thread T1: { down(s1); print("A"); down(s2); print("B"); up(s2); up(s1); }
Thread T2: { down(s2); down(s1); print("C"); up(s2); up(s1); }

Quando il programma viene eseguito
1. stampa sempre al più due lettere (motivare la risposta)
Falso può stampare 3 lettere

2. presenta problemi di deadlock (motivare la risposta)
No perchè s1=2 non è bloccante su due "down"

3. stampa sempre tre lettere (motivare la risposta)
Si se si considera scheduling "fair"

4. può intervallare istruzioni di T1 e T2 (motivare la risposta)
Si non ci sono esecuzioni in cui istruzioni risultano bloccate

D8 (4 punti) Confrontare (con brevi spiegazioni) il comportamento dei seguenti programmi C

Programma A

```
int idx=0, idy=0;

void *foo(void *aux) {
  idx=20;
  idy=40;
  printf("Thread %d\n", idx);
  return(0);
}

int main() {
   pthread_t tid;
   void *ret;
   pthread_create(&tid, 0, foo, 0);
   idx=100;
   idy=100;
   pthread_join(tid,&ret);
   return 0;
}
```

Programma B

```
int idx=0, idy=0;

void *foo(void *aux) {
  idx=20;
  idy=40;
  return(0);
}

int main() {
   pthread_t tid;
   void *ret;
   pthread_create(&tid, 0, foo, 0);
   idx=100;
   idy=100;
   pthread_join(tid,&ret);
   printf("Thread %d\n", idx);
   return 0;
}
```

Risposta:

Entrambi i programmi possono stampare sia 100 che 20

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

```
class mydata{
   private a=0;
   public synchronized m1(){ a=a+1; }
   public m2(){ a=3;  }
   public synchronized m3(){ print(a);  }
}
```

Thread 1 (A): A.m1(); A.m1();
Thread 2 (A): A.m2(); A.m3();

Main:
mydata A = new mydata();
creazione con reference ad A ed avvio di un'istanza di Thread 1;
creazione con reference ad A ed avvio di un'istanza di Thread 2;

1. Il programma può stampare 5? (motivare la risposta)

Si es attraverso la computazione A.m1() A.m1() A.m2() A.m3()

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

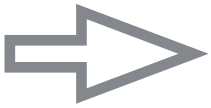
Non in computazioni fair perchè ogni sezione critica non contiene istruzioni bloccanti

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

Si perchè m2 non è sincronizzato e se m1 e m2 vengono eseguiti in parallelo m1 può cancellare assegnamento a=3

4. Il programma può stampare 2? (motivare la risposta)

Si perchè m2 non è sincronizzato e se m1 e m2 vengono eseguiti in parallelo m1 può cancellare assegnamento a=3, eseguendo ancora m1 e poi m3 il risultato sarà 2



D11 (8 punti) Un sistema integrato è in grado di attivare simultaneamente la produzione di al più 5 prototipi. Un singolo prototipo viene creato tramite l'operazione crea_prototipo(file) dove file contiene il progetto dell'oggetto. Per un errore nella costruzione degli apparati, quando si superano le 5 richieste simultanee l'esecuzione di crea_prototipo(file) manda in errore il sistema con conseguente possibili guasti alle macchine.

Scrivere lo schema di un server multithreaded in grado di gestire richieste multiple al sistema integrato in maniera che non vada mai in errore.

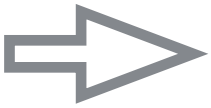
Risposta:

Tra le varie possibilità:

- Algoritmo per un server multithreaded tradizionale (vedi lucidi) con un semaforo/monitor a protezione delle chiamate concorrenti a crea_prototipo con un contatore per controllare il numero di thread in sezione critica (o un semaforo inizializzato a 5)

- Algoritmo per un server multithreaded tradizionale (vedi lucidi) con un pool con al massimo 5 threa dove eseguire le chiamate concorrenti a crea_prototipo (o un semaforo inizializzato a 5)

Lo schema proposto deve chiarire come il server gestisce una richiesta ricevuta da un client attraverso un thread (client virtuale), in altre parole bisogna descrivere l'algoritmo con il quale opera il server



D12 (6 punti)
Scrivere un esempio di algoritmo lock-free per risolvere il problema della sezione critica.

Risposta:

Abbiamo visto numerosi esempi:
soluzioni software alla sezione critica, soluzioni con test-and-set e swap-and-compare