

# Elementi di Architettura e Sistemi Operativi

Bioinformatica - Tiziano Villa

24 Febbraio 2017

Nome e Cognome:

Matricola:

Posta elettronica:

problema	punti massimi	i tuoi punti
problema 1	9	
problema 2	6	
problema 3	5	
problema 4	10	
totale	30	

1. Si risponda in modo preciso e conciso alle seguenti domande.

(a) Che cos'è una macchina virtuale ?

Traccia di soluzione.

Una macchina virtuale è un'emulazione in software di una macchina astratta. Si può usare, ad esempio, per permettere la coesistenza di più sistemi operativi sulla medesima architettura, o per proteggere il sistema operativo sottostante, o per presentare un ambiente di esecuzione virtuale.

(b) Una dipendenza ciclica nella richiesta di risorse porta sempre a uno stallo dei processi ?

Traccia di soluzione.

No. È una condizione necessaria, ma non sufficiente quando ci sono risorse con più unità (è necessaria e sufficiente per risorse con una sola unità). Il motivo è che un processo che non fa parte del ciclo potrebbe rendere disponibile un'unità di una risorsa richiesta da un processo che è nel ciclo, con l'effetto d'interrompere il ciclo.

(c) Che cosa sono le eccezioni ? Che cosa sono le eccezioni sincrone (si diano esempi) ? Che cosa sono le eccezioni asincrone (si diano esempi) ?

Traccia di soluzione.

Le eccezioni sono eventi che fermano l'esecuzione normale. e fanno passare dalla modalità utente a quella di sistema attivando una procedura corrispondente. Le eccezioni possono essere sincrone o asincrone. Esempi di eccezioni sincrone sono le chiamate di sistema, la divisione per zero, le istruzioni illegali, e le mancanze di pagine. Esempi di eccezioni asincrone sono le interruzioni (da orologio per tempo scaduto, da rete o da disco).

(d) Si menzionino i possibili svantaggi quando si decompone un processo in molti (troppi) flussi esecutivi ("threads").

Traccia di soluzione.

La resa diminuisce perché aumenta il numero di cambi di contesto.

Aumento dell'uso di spazio per gestire i blocchi di controllo (TCB) e altre strutture dati relative.

Costi per la gestione della sincronizzazione quando si divide un problema in un numero crescente di processi.

Difficolta' di realizzare correttamente la sincronizzazione, con rischi di esecuzioni errate o blocco di processi.

- (e) i. Un processo puo' passare da *bloccato* a *in esecuzione* ?

Traccia di soluzione.

Non e' legale nella maggior parte dei sistemi operativi. Si sceglie che cosa eseguire dalla lista dei processi nello stato *pronto per l'esecuzione*. Se si usano monitors nello stile di Hoare, si potrebbe passare da essere *bloccato* (ad es., su un lucchetto) direttamente a *in esecuzione*.

- ii. Un processo puo' passare da *in esecuzione* a *bloccato* ?

Traccia di soluzione.

Si. Ad esempio, quando chiede una risorsa (es., accesso al disco) o un lucchetto, si sincronizza con una *join*, etc.

- iii. Un processo puo' passare da *pronto per l'esecuzione* a *bloccato* ?

Traccia di soluzione.

Non e' legale: si passa nello stato di *bloccato* solo da *in esecuzione*, cioe' si deve eseguire un'azione che causa il blocco e quindi bisogna essere *in esecuzione*.

2. Si consideri il seguente paradigma di sincronizzazione, nel caso di due processi che competono per entrare in una sezione critica. Si assuma che:

- (a) Una sezione critica e' protetta se vi puo' accedere un solo processo per volta;
- (b) la sincronizzazione e' equa se ogni processo ha le medesima possibilita' di accedervi.
- (c) All'inizio vale

`flag = false;`

Si supponga che ognuno dei due processi esegua il codice seguente:

```
1. while (flag == true)
2. do nothing;
3. flag = true;
4. si entra nella sezione critica;
5. flag = false;
```

Si risponda alle seguenti domande.

- (a) Questo meccanismo di sincronizzazione garantisce la protezione della sezione critica ?

Se si, si argomenti il motivo. Se no, si dia un esempio di esecuzione dei due processi che viola la protezione.

Traccia di soluzione.

Il codice precedente funziona il piu' delle volte: arriva un processo *A* che trova *flag* falso, salta il corpo del while, mette *flag* a vero e poi entra nella sezione critica. Poi arriva un processo *B*, ma e' fermato dalla guardia del ciclo while perche' adesso *flag* e' vero e tale rimane fino a che il processo *A* esce dalla sezione critica mettendo *flag* a falso. Allora la risposta e' SI ?

No, la risposta e' NO, perche' ci sono delle situazioni in cui le cose non procedono come sopra. Si supponga che *A* esegua la riga 1., verifichi che la guardia del predicato e' falsa, per cui salta il corpo del ciclo while, e poi che sia interrotto prima di passare alla riga 3 (perche' interrotto ? per vari motivi, magari e' scaduto il suo quanto di tempo). Adesso si supponga che *B* esegua a sua volta la riga 1., verifichi anch'esso che la

guardia del predicato e' falsa, per cui salta anch'esso il corpo del ciclo while. Adesso sia  $A$  che  $B$  possono entrare nella sezione critica.

(b) Questo meccanismo di sincronizzazione e' equo ?

Traccia di soluzione.

Sì. Il codice e' simmetrico, perciò ogni processo ha le medesime possibilità di accedervi.

3. L'insieme delle istruzioni della macchina LC-3 non ne prevede una per realizzare la funzione logica disgiunzione (OR). Tuttavia si può simulare l'operazione OR usando altre istruzioni disponibili. Sotto s'indica una successione incompleta di 4 istruzioni che eseguono la disgiunzione dei contenuti dei registri R1 e R2 e salvano il risultato nel registro R3. Si scrivano le due istruzioni mancanti (nel formato binario) in modo che le quattro istruzioni eseguano l'operazione OR. Si commenti ogni istruzione, spiegando che cosa fa.

(a) (1) 1001 100 001 111111

(b) (2)

(c) (3) 0101 110 100 000 101

(d) (4)

Traccia di soluzione.

(a) (1) 1001 100 001 111111

Nega il contenuto del registro R1 e lo salva nel registro R4

(b) (2) 1001 101 010 111111

Nega il contenuto del registro R2 e lo salva nel registro R5

(c) (3) 0101 110 100 000 101

Esegue il prodotto logico (AND) dei contenuti dei registri R4 e R5 e lo salva nel registro R6

(d) (4) : 1001 011 110 111111

Nega il contenuto del registro R6 e lo salva nel registro R3.

La sequenza d'istruzioni esegue la disgiunzione logica grazie alle seguenti identità dell'algebra di Boole:

$$R1 + R2 = \overline{\overline{R1} + \overline{R2}} = \overline{\overline{R1} \wedge \overline{R2}}.$$

4. Si progetti un circuito sequenziale contatore binario su 2 cifre binarie, con due segnali binari d'ingresso  $C$  e  $D$ , con le seguenti caratteristiche:
- (a) il contatore aggiorna il conteggio (in avanti o all'indietro) oppure mantiene il valore corrente in base al valore di un segnale d'ingresso  $C$  (se  $C = 0$  mantiene il conteggio corrente, se  $C = 1$  aggiorna il conteggio);
  - (b) il contatore conta in avanti oppure all'indietro in base al valore di un segnale d'ingresso  $D$  (se  $D = 0$  conta in avanti, se  $D = 1$  conta all'indietro).

Le uscite che rappresentano il conteggio coincidono con le variabili di stato.

In particolare si svolgano i seguenti passi:

- (a) Si progetti la macchina a stati finiti della specifica (tipo Moore).
- (b) Si scriva la tavola delle transizioni e la si codifichi.
- (c) Si scrivano le equazioni minimizzate della logica che genera lo stato futuro e le uscite.
- (d) Si disegni lo schematico del circuito sequenziale con porte logiche e bistabili di tipo D.

Traccia di soluzione.

Si rinvia alle dispense dove si trova tale esempio.

Si noti che le uscite coincidono con le variabili di stato presente, quindi al riguardo basta derivare le uscite dagli stati presenti dei bistabili (che sono anche retroazionati come ingressi alla logica combinatoria di stato futuro per calcolare gl'ingressi dei bistabili).