

# Elementi di Architettura e Sistemi Operativi

Bioinformatica - Tiziano Villa

6 Settembre 2013

Nome e Cognome:

Matricola:

Posta elettronica:

problema	punti massimi	i tuoi punti
problema 1	7	
problema 2	7	
problema 3	6	
problema 4	10	
totale	30	

1. Si propone una nuova tecnica di sincronizzazione che si basa sul meccanismo del numero di coda in un negozio o ufficio (l'utente arriva e prende il numero da una macchinetta emettitrice e poi aspetta fino a che il suo numero e' chiamato, potendo controllare da uno schermo qual e' il numero servito correntemente).

In particolare, s'introducono le variabili *numero\_servito* e *numero\_arrivo*.

Per *numero\_servito* s'intende una variabile intera rispetto a cui sono definite tre operazioni:

- (a) *inizializza\_servito(numero\_servito)*, azzera il valore di *numero\_servito*;
- (b) *incrementa\_servito(numero\_servito)*, incrementa atomicamente il valore di *numero\_servito*;
- (c) *aspetta\_servito(numero\_servito, valore)*, fa attendere il processo chiamante fino a che il valore di *numero\_servito* e' diventato maggiore o uguale al *valore* in argomento.

Per *numero\_arrivo* s'intende una variabile intera rispetto a cui sono definite due operazioni:

- (a) *inizializza\_arrivo(numero\_arrivo)*, azzera il valore di *numero\_arrivo*;
- (b) *incrementa\_arrivo(numero\_arrivo)*, incrementa atomicamente il valore di *numero\_arrivo* e restituisce il suo valore precedente (cioe' modella la macchinetta emettitrice che assegna all'utente il numero corrente e lo incrementa per il prossimo utente).

Supponendo di avere a disposizione le operazioni precedenti, si scrivano tre procedure *Inizializza\_semaforo(sema, valore)*, *P(sema)*, *V(sema)*, che usano una variabile di tipo *numero\_servito* e una di tipo *numero\_arrivo* per realizzare un semaforo. *Inizializza\_semaforo* assegna un valore iniziale al semaforo, *P* e *V* sono le classiche operazioni sui semafori. **S'illustri con chiarezza il codice proposto.**

Si assuma che *sema* sia un puntatore a una struttura che contiene *numero\_servito* e *numero\_arrivo*, essendo tale struttura definita come segue

```
typedef struct {
    int numero_servito;
    int numero_arrivo;
} semaforo;
```

Traccia per lo studente.

```
Inizializza_semaforo(semaforo *sema, int valore) {
    int fittizio;
    inizializza_servito(sema->numero_servito);
    inizializza_arrivo(sema->numero_arrivo);

    /* si simula un utente fittizio per iniziare
       a contare gli utenti da 1 e non da 0 */
    fittizio = incrementa_arrivo(sema->numero_arrivo);

    /* si inizializza numero_servito a valore */
    while (valore-- > 0)
        incrementa_servito(sema->numero_servito))
}
```

```
P{?) {
    ?

    ? = ?;
    ?;
}
```

```
V{?) {

    ?;
}
```

Traccia di soluzione.

Osservazioni preliminari:

- Le variabili *numero\_servito* e *numero\_arrivo* possono soltanto essere incrementate, ma non possono essere lette. Si può ottenere soltanto il valore di *numero\_arrivo* (tramite *incrementa\_arrivo*, che restituisce il valore di *numero\_arrivo* prima d'incrementarlo).
- La funzione *aspetta\_servito* realizza una forma d'attesa e quindi ha a che fare con l'operazione semaforica *P*.
- La funzione *incrementa\_servito* ha a che fare con l'operazione semaforica *V* perché incrementa una variabile su cui un processo può essere in attesa.

Per realizzare un semaforo, si userà *numero\_servito* per contare quante *V* sono state eseguite sul semaforo, e *numero\_arrivo* per contare quante *P* sono state eseguite sul semaforo. Quando un processo esegue *V* su un semaforo, incrementa il valore di *numero\_servito*. Quando un processo esegue *P* su un semaforo, riceve il valore corrente di *numero\_arrivo*, che indica al processo il suo numero d'ordine nella coda semaforica (e lo incrementa perché sia pronto per la prossima *P*); se  $\text{numero\_servito} < \text{numero\_arrivo}$ , il processo deve aspettare che *numero\_servito* s'incrementi fino al valore di *numero\_arrivo* ricevuto, altrimenti il processo può proseguire. Il valore del semaforo è  $\text{numero\_servito} - \text{numero\_arrivo} + 1$ .

```

Inizializza_semaforo(semaforo *sema, int valore) {
    int fittizio;
    inizializza_servito(sema->numero_servito);
    inizializza_arrivo(sema->numero_arrivo);

    /* si simula un utente fittizio per iniziare
       a contare gli utenti da 1 e non da 0 */
    fittizio = incrementa_arrivo(sema->numero_arrivo);

    /* si inizializza numero_servito a valore */
    while (valore-- > 0)
        incrementa_servito(sema->numero_servito))
}

P(semaforo *sema) {
    int numero_coda;

    numero_coda = incrementa_arrivo(sema->numero_arrivo);
    aspetta_servito(sema->numero_servito, numero_coda);
}

V(semaforo *sema) {
    incrementa_servito(sema->numero_servito);
}

```

Si noti che l'esecuzione di una *P* consiste in

- *incrementa\_arrivo*: prende un nuovo biglietto dalla emettitrice (e incrementa tale numero per l'utente successivo)
- *aspetta\_servito*: confronta tale numero con il valore di *numero\_servito*, se tale numero è maggiore di *numero\_servito* devo aspettare il mio turno (cioè ho un numero di coda maggiore di quello dell'utente servito adesso, per cui devo aspettare che arrivi il turno del mio numero).

2. (a) Si descriva l'algoritmo di schedulazione del processore denominato schedulazione per brevità (SJF, il processo più breve esegue per primo).

In particolare, si descriva il metodo per determinare la lunghezza della sequenza successiva di operazioni del processore basato sul calcolo della media esponenziale delle durate effettive delle sequenze precedenti di operazioni del processore.

Traccia di soluzione.

Si veda la sezione relativa nel libro di testo.

(b) Si consideri la formula della media esponenziale atta a predire la durata della sequenza successiva del processore.

Quali implicazioni scaturiscono dall'assegnazione dei seguenti valori ai parametri della formula ?

i.  $\alpha = 0; \tau_0 = 100 \text{ millisecondi}$

ii.  $\alpha = 0,99; \tau_0 = 10 \text{ millisecondi}$

Traccia di soluzione.

Formula della media esponenziale:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

dove  $t_n$  e' la lunghezza dell' $n$ -esima sequenza di operazioni del processore,  $\tau_{n+1}$  e' il valore previsto per la sequenza successiva di operazioni del processore,  $0 \leq \alpha \leq 1$ . In breve,  $t_n$  contiene le informazioni piu' recenti,  $\tau_n$  registra la storia passata.

Se  $\alpha = 0$ ,  $\tau_{n+1} = \tau_n = \dots = \tau_0 = 100 \text{ millisecondi}$ , cioe' la formula predice sempre 100 come durata del prossimo utilizzo del processore, e la storia recente non ha effetto (si suppone che le condizioni attuali siano transitorie).

Se  $\alpha = 0,99$ , il comportamento piu' recente del processore conta molto di piu' della sua storia passata, quindi l'algoritmo di schedulazione e' quasi senza memoria e predice una durata del prossimo accesso al processore quasi pari a quella dell'ultimo accesso.

3. (a) Si spieghi il meccanismo della paginazione nella memoria centrale.

Traccia di soluzione.

Si veda la sezione relativa nel libro di testo.



- (b) Si spieghi perché è più facile condividere un modulo di codice rientrante usando la segmentazione anziché la paginazione pura (si definisca che cos'è un codice rientrante).

Traccia di soluzione.

Segmenti di qualsiasi dimensione possono essere condivisi con un solo elemento nelle tavole dei segmenti di ciascun processo. Con la paginazione ci deve essere un elemento in comune nelle tavole delle pagine per ogni pagina condivisa.

- (c) Assumendo la dimensione di pagina di 1 KB quali sono i numeri di pagina e gli scostamenti per i seguenti indirizzi (indicati in numeri decimali):

- i. 2375
- ii. 19366
- iii. 30000
- iv. 256
- v. 16385

Traccia di soluzione.

- i. pagina = 2, scostamento = 327 ( $2375 \% 1024 = 327$ ;  $2375 / 1024 = 2$ )
- ii. pagina = 18, scostamento = 934 ( $19366 \% 1024 = 934$ ;  $19366 / 1024 = 18$ )

- iii. pagina = 29, scostamento = 304 ( $30000 \% 1024 = 304$ ;  $30000 / 1024 = 29$ )
- iv. pagina = 0, scostamento = 256 ( $256 \% 1024 = 256$ ;  $256 / 1024 = 0$ )
- v. pagina = 16, scostamento = 1 ( $16385 \% 1024 = 1$ ;  $16385 / 1024 = 16$ )

4. Si progetti un circuito sequenziale che realizza la seguente specifica:

- Ci sono un segnale binario d'ingresso  $X$  e un segnale binario d'uscita  $Z$ .
- L'uscita  $Z$  vale 1 se su  $X$  si e' presentata una successione di un numero pari di 0 seguiti da un numero dispari di 1; l'uscita  $Z$  torna a 0 quando il numero di 1 diventa pari (per tornare a 1 se torna dispari) o si ripresenta in ingresso uno 0.

(a) Si disegni il grafo delle transizioni di una macchina a stati finiti di tipo Mealy che corrisponde alla specifica. S'indichi lo stato iniziale.

- (b) Si minimizzi il numero degli stati della macchina proposta, applicando l'algoritmo di minimizzazione degli stati.

- (c) Si scriva la tavola delle transizioni con gli stati futuri e le uscite e la si codifichi.

- (d) Supponendo di usare bistabili di tipo D, si derivino le equazioni minimizzate di eccitazione degl'ingressi dei bistabili e le equazioni minimizzate delle uscite.

- (e) Si realizzi il circuito sequenziale corrispondente con bistabili di tipo D campionati sul fronte di salita, invertitori e porte NAND. Si etichettino con chiarezza i segnali.