Corso di Laurea in Informatica, A.A. 2022/2023

Architetture degli Elaboratori e Sistemi Operativi (AESO corso A e B)

Sesto Appello – 11 Gennaio 2024

Scrivere in modo comprensibile e chiaro rispondendo alle domande/esercizi riportati in questo foglio. Sviluppare le soluzioni dei primi due esercizi (prima parte) in un foglio separato rispetto alla soluzione degli ultimi due esercizi (seconda parte), in modo da facilitare la correzione da parte dei docenti. Riportare su tutti i fogli consegnati nome, cognome, numero di matricola e corso (A o B).

Parte 1

Esercizio 1

Supponiamo di avere il tipo `struct elem {int k; char *s;};` ovvero una struttura contenente un intero chiave *k* e un puntatore ad una stringa *s*. Scrivere l'implementazione in ARMv7 della funzione con firma seguente

```
int lenOccorrenze(struct elem *v, int N, int key);
```

La funzione prende come primo parametro un vettore di elementi di tipo *struct elem*, il numero $N \geq 0$ di elementi di questo vettore, e un intero *key*. Il risultato della funzione è un intero che rappresenta la somma delle lunghezze di tutte le stringhe che sono associate nel vettore alla chiave key fornita come parametro.

Non è ammesso utilizzare strlen o qualsiasi altra funzione della libreria libc.

Esercizio 2

Si progetti una rete sequenziale con ingressi *A* e *B* a N bit ciascuno, e *op0* e *op1* a 1 bit ciascuno. Il comportamento della rete è descritto dallo pseudocodice seguente, dove il registro *Z* a N bit è il registro che implementa lo stato interno

```
if(Cond(Z))then
begin
    switch(op1, op0) {
        case 00 -> Z = Z + A + B;
        case 01 -> Z = Z + A - B;
        case 10 -> Z = A;
        case 11 -> Z = 0;
    }
end
else nop;
```

La rete va progettata utilizzando componenti standard, assumendo che venga eseguita l'operazione descritta dallo pseudocodice ad ogni ciclo di clock e utilizzando non più di un multiplexer. Si assuma di avere a disposizione solamente un componente a due livelli di logica che calcola la funzione *Cond()*, **un solo** multiplexer, porte and/or/not, e un numero arbitrario di full adder operanti a N bit con un ritardo pari al tempo di stabilizzazione di otto livelli di logica ciascuno. Si indichi un ragionevole lower bound alla lunghezza del ciclo di clock necessario a far funzionare correttamente la rete sequenziale.

Opzionale: risolvere l'esercizio senza il vincolo dell'utilizzo di un solo multiplexer.

Parte 2

Esercizio 3

a) Descrivere brevemente come si realizza e le principali problematiche dell'algoritmo MFQ per sistemi multiprocessore implementato con un unico set di code pronti condivise da tutti i processori del sistema (implementazione centralizzata).

b) Un sistema schedula i thread implementati a livello kernel con la tecnica MFQ (Multilevel Feedback Queue). La politica di scheduling prevede il prerilascio. Quando un thread va in esecuzione gli viene assegnato un intero quanto di tempo pari a 10 ms, indipendentemente dal tempo consumato nel precedente turno di esecuzione. La priorità di un thread viene incrementata di 1, fino ad arrivare al valore massimo 3, ogni volta che il thread viene deschedulato senza aver consumato per intero il quanto di tempo. La priorità del thread viene invece decrementata di 1, fino al valore minimo di 1, ogni volta che il thread consuma per intero il suo quanto di tempo. Le code dei semafori e delle mutex sono code a priorità, per pari priorità l'ordine di accodamento è FIFO.

Al tempo *t* sono presenti nel sistema i seguenti thread:

- T1, con priorità 3, che al tempo *t* passa in stato *esecuzione*;
- T2, con priorità 2, che al tempo *t* è in stato di *attesa* sul semaforo *Sem1*;
- T3, con priorità 2, che al tempo *t* è in stato di *attesa* sulla lock L;
- T4, con priorità 3, che al tempo *t* è in stato di *pronto*;
- T5, con priorità 1, che al tempo *t* è in stato di *pronto*.

Si chiede qual è il thread in esecuzione e lo stato delle 3 code pronti e delle variabili L, Sem1, ed SL al termine della seguente sequenza di eventi:

- al tempo *t+2* il thread in esecuzione esegue V(Sem1);
- al tempo *t+4* il thread in esecuzione esegue V(Sem1);
- al tempo *t+8* il thread in esecuzione esegue spinLock(SL);
- al tempo *t+16* il thread in esecuzione esegue Lock(L);
- al tempo *t+28* il thread in esecuzione esegue spinLock(SL);
- al tempo *t+30* il thread in esecuzione termina;
- al tempo *t+32* il thread in esecuzione esegue Unlock(L);
- al tempo *t+34* il thread in esecuzione esegue P(Sem1);
- al tempo t+ 44 il thread in esecuzione esegue Unlock(L);
- al tempo t+ 46 il thread in esecuzione termina.
- al tempo t+ 48 il thread in esecuzione esegue spinUnlock(SL);
- al tempo t+ 50 il thread in esecuzione esegue Unlock(L);
- al tempo t+ 60 il thread in esecuzione termina

La soluzione deve essere fornita completando la seguente tabella (T1(3) indica che il thread T1 ha priorità 3) aggiungendo il numero di righe necessarie.

t+	**Evento**	**Subito dopo l'evento**						
		Exec	**Coda prio 1**	**Coda prio 2**	**Coda prio 3**	**Sem1 val, coda**	**Lock val, coda**	**SL val**
0	T1 parte	T1(3)						

Esercizio 4

a) Descrivere brevemente le caratteristiche principali dell'algoritmo del Banchiere.

b) In un sistema Unix sono presenti i Pthread T1, T2 e T3 che comunicano tramite due code Q1 ed Q2 di tipo *coda_t*. Il Pthread T1 effettua un ciclo nel quale produce una sequenza infinita di numeri interi e li deposita in Q1 uno alla volta. Il Pthread T2 effettua un ciclo nel quale preleva due valori da Q1 e deposita la loro somma in Q2. Il Pthread T3 effettua un ciclo infinito nel quale preleva un valore da Q2 e lo consuma. La massima size delle code Q1 e Q2 è rispettivamente pari ad N>1 ed M>0. Si supponga che il tipo di dato coda_t sia già stato implementato e che siano disponibili i seguenti metodi non sincronizzati:

- int *push*(coda_t* Q, int v) inserisce l'intero v nella coda Q; se l'operazione è andata a buon fine ritorna 0 altrimenti -1;
- int *pop*(coda_t* Q, int* v) preleva il primo elemento della coda e lo memorizza in v; se l'operazione è andata a buon fine ritorna 0 altrimenti -1;
- int *length*(coda_t* Q) ritorna il numero di elementi nella coda Q.

Si chiede di implementare il codice in C dei Pthread T1,T2, e T3 utilizzando per la sincronizzazione variabili di *mutex* (pthread_mutex_t) e variabili di condizione (pthread_cond_t) in modo appropriato.

SOLUZIONE

Eserizio 1

```
    .text
    .global lenOccorrenze
    .type lenOccorrenze, %function

lenOccorrenze:       @ r0 vet, r1 n, r2 k
    push {r4}        @salvataggio non temporanei
    mov r4, r0       @ uso r4 per vet, r1 e r2 rimangono N e k
    mov r12, #0      @ l (retval)    (in un temporaneo)

loop:
    cmp r1, #0            @ controllo e ho finito gli elementi del vettore
    beq finito            @ e in caso esco
    ldr r3, [r4]                @ carica chiave dell'elemento corrente
    cmp r2,r3             @ vedo se è quella che cerco
    bne fineloop          @ se non è quella cercata fai un altro giro

    ldr r0, [r4,#4]       @ carico indirizzo della stringa
conta:
    ldrb r3, [r0], #1     @ carico carattere corrente
    cmp r3, #0            @ è un null?
    beq fconta            @ esci dalla conta
    add r12, r12, #1      @ n++
    b conta               @ e vai al prossimo carattere
fconta:                   @ finita la stringa, prossima iter vet
fineloop:
    add r4, r4, #8        @  prossimo elemento
    sub r1, r1, #1        @  uno in meno da controllare
    b loop

finito:
    mov r0, r12      @ valore da restiuire in R0
    pop {r4}         @ ripristino registri
    mov pc, lr       @ e ritorno
```

Esercizio 2

Si utilizzano due full adder per calcolare A+B e A-B (come A + not(B) + 1). Le uscite dei due full adder alimentano il primo ingresso di altri due full adder che hanno come secondo ingresso l'uscita di Z. L'uscita di questi due full adder, insieme ad A e alla costante 0 sono ingressi di un multiplexer a 4 vie che genera il segnale in ingresso a Z. L'ingresso di controllo del multiplexer è dato dai due bit op. Il segnale di write enable di Z è generato dalla rete che calcola Cond() utilizza con ingresso l'uscita di Z.

Dal momento in cui il valore dell'uscita di Z è stabile occorre attendere la stabilizzazione a cascata di due full adder e del multiplexer prima di passare al prossimo ciclo di clock. La stabilizzazione di Cond può invece avvenire in parallelo. Quindi il ciclo di clock deve essere lungo almeno quanto 2 Tsum + Tmux (con le assunzioni date si tratta di 18 livelli di logica).

Opzionale: qualora venga rimosso il vincolo del singolo multiplexer, si può utilizzare un unico full adder per calcolare sia A+B che A-B (come nella struttura della ALU del libro), ma poi serve comunque un secondo full adder per sommare con Z e un multiplexer da 3 ingressi (in realtà 4, essendo la potenza di 2 più vicina) per scegliere fra il risultato del secondo full adder e le costanti A e 0. Il tempo di stabilizzazione deve tener conto che occorre sommare il ritardo di un secondo Tmux per la scelta fra B e not(B) sul secondo ingresso del primo full adder. Il lower bound al ciclo di clock è quindi di 20 livelli di logica. Dunque il ciclo di clock è pià lungo e si utilizzano un full adder in meno e un multiplexer a 2 vie in più.

Esercizio 3

Notazione: T1->T2 vuol dire che T1 viene prima di T2. T1(3) indica che il thread T1 ha priorità 3

t+	Evento	Subito dopo l'evento						
		Exec	**Coda prio 1**	**Coda prio 2**	**Coda prio 3**	**Sem1 val, coda**	**Lock val, coda**	**SL val**
0	T1 parte	T1(3)	T5	-	T4	0,T2(2)	busy, T3(2)	busy
2	V(Sem1)	T1(3)	T5	T2	T4	0,-	busy, T3(2)	busy
4	V(Sem1)	T1(3)	T5	T2	T4	1,-	busy, T3(2)	busy
8	spinLock(SL)	T1(3)	T5	T2	T4	1,-	busy, T3(2)	busy
10	Scade QdT	T4(3)	T5	T2->T1	-	1,-	busy, T3(2)	busy
16	Lock(L)	T2(2)	T5	T1	-	1, -	busy,T4(3)->T3(2)	busy
26	Scade QdT	T1(2)	T5->T2	-	-	1, -	busy,T4(3)->T3(2)	busy
28	spinLock(SL)	T1(2)	T5->T2	-	-	1,-	busy, T4(3)->T3(2)	busy
30	T1 termina	T5(1)	T2	-	-	1,-	busy, T4(3)->T3(2)	busy
32	Unlock(L)	T4(3)	T2	T5	-	1,-	busy, T3(2)	busy
34	P(Sem1)	T4(3)	T2	T5	-	0, -	busy, T3(2)	busy
42	Scade QdT	T5(2)	T2	T4	-	0, -	busy, T3(2)	busy
44	Unlock(L)	T5(2)	T2	T4->T3	-	1,-	busy,-	busy
46	T5 termina	T4(2)	T2	T3	-	1,-	busy,-	busy
48	spinUnlock(SL)	T4(2)	T2	T3	-	1,-	busy,-	free
50	Unlock(L)	T4(2)	T2	T3	-	1,-	free,-	free
56	Scade QdT	T3(2)	T2->T4	-	-	1,-	free,-	free
60	T3 termina	T2	T4	-	-	1,-	free,-	free

Esercizio 4

```
pthread_mutex_t mutexQ1     = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexQ2     = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condQ1Full   = PTHEAD_COND_INITIALIZER;
pthread_cond_t condQ1Empty  = PTHREAD_COND_INITIALIZER;
pthread_cond_t condQ2Full   = PTHEAD_COND_INITIALIZER;
pthread_cond_t condQ2Empty  = PTHREAD_COND_INITIALIZER;
```

Pthread T1:

```
for(;;) {
      <produce un intero d>
      pthread_mutex_lock(&mutexQ1);
      while (length(&Q1)==N) pthread_cond_wait(&condQ1Full,&mutexQ1);
      push(Q1, d); // l'operazione sicuramente va a buon fine
      if (length(&Q1) == 2) pthread_cond_signal(&condQ1Empty);
      pthread_mutex_unlock(&mutexQ1);
}
```

Pthread T2:

```
for(;;) {
      pthread_mutex_lock(&mutexQ1);
      while (length(&Q1) < 2) pthread_cond_wait(&condQ1Empty,&mutexQ1);
      int d1, d2;
      pop(Q1, &d1); // l'operazione sicuramente va a buon fine
      pop(Q1, &d2); // l'operazione sicuramente va a buon fine
      pthread_cond_signal(&condQ1Full);
      pthread_mutex_unlock(&mutexQ1);
      pthread_mutex_lock(&mutexQ2);
      while (length(&Q2) == M) pthread_cond_wait(&condQ2Full,&mutexQ2);
      int d = d1+d2;
      push(Q2, d); // l'operazione sicuramente va a buon fine
      pthread_cond_signal(&condQ2Empty);
      pthread_mutex_unlock(&mutexQ2);
}
```

Pthread T3:

```
for(;;) {
      pthread_mutex_lock(&mutexQ2);
      while (length(&Q2) == 0) pthread_cond_wait(&condQ2Empty,&mutexQ2);
      int s;
      pop(Q2, &s); // l'operazione sicuramente va a buon fine
      pthread_cond_signal(&condQ2Full);
      pthread_mutex_unlock(&mutexQ2);
      <usa s>
}
```