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

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

Quarto Appello – 4 Settembre 2023

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

Utilizzando l'assembler ARMv7 visto a lezione, implementare la funzione che calcola la parte intera della radice quadrata di un numero passato come parametro. L'algoritmo da utilizzare comincia a considerare come soluzione 1 e prosegue incrementando la soluzione di 1 fino a quando il quadrato di (soluzione + 1) non diventi maggiore del numero passato come parametro. Quando è maggiore, si restituisce direttamente il valore di soluzione.

Esercizio 2

Si progetti un componente logico che, dati gli ingressi x, y e k, rispettivamente da 8, 8 e 3 bit, restituisce in uscita z, da 8 bit, calcolata secondo il seguente algoritmo:

```
if (x == 2^k)
  z= y+x;
else
  z= x + x;
```

Del componente logico si fornisca il ritardo di stabilizzazione in Δt, assumendo di avere a disposizione porte da 8 ingressi che stabilizzano in un Δt.

Parte 2

Esercizio 3

a) Si descrivano brevemente le condizioni necessarie per avere stallo/deadlock.

b) Assumendo di avere a disposizione un tipo di dato **vettore_t** di capacità infinita con operazioni di **void push_back(vettore_t v, msg_t messaggio)** che inserisce il messaggio in ultima posizione nel vettore *v*, **void pop_front(vettore_t v, msg_t *msg)** che elimina il primo elemento del vettore restituendo in msg il messaggio, e **int length(vettore_t v)** che ritorna il numero di elementi attualmente nel vettore, si fornisca lo pseudo codice della procedura **void send(vettore_t v, msg_t msg)** e della procedura **void receive(vettore_t vettore, msg_t *msg)** che utilizzano tale vettore come buffer di comunicazione per un canale di comunicazione *multi-producer/multi-consumer*, utilizzando, ove necessario, variabili di mutua esclusione e di condizione con semantica MESA per realizzare la sincronizzazione fra i produttori ed i consumatori.

Considerare, ora, il caso in cui il vettore di tipo **vettore_t** abbia capacità finita *k*. Fornire l'implementazione di **send** e **receive** utilizzando una sola variabile di condizione.

Esercizio 4

a) Si descriva brevemente in cosa consiste una politica di schedulazione a quanto di tempo.

b) Assumendo di operare con una politica di schedulazione "SJF speciale" descritta qui di seguito, si indichi l'ordine di esecuzione deciso dallo schedulatore ed il tempo di permanenza medio dei processi riportati in tabella. L'ordine di arrivo è quello lessicografico (prima A, poi B, poi C, ...). Assumiamo, per rendere i conti più semplici, che tutti i processi siano arrivati in un brevissimo intervallo di tempo intorno al tempo 0 e che *t* sia la lunghezza del quanto di tempo assegnato dallo schedulatore ad ogni processo. Assumiamo, ancora per semplicità, che nessuno dei processi richieda operazioni di ingresso/uscita. Infine, lo schedulatore, a parità di tempo residuo sceglie sempre il processo da schedulare in ordine lessicografico.

SJF speciale: va in esecuzione, per un quanto di tempo, il task con tempo residuo minore di tutti, salvo che esista un altro task con tempo residuo inferiore o uguale a due quanti di tempo e che nelle ultime due operazioni di scheduling sia sempre stato scelto il task con tempo residuo minore.

Proc	Tempo di esecuzione
A	2t
B	1t
C	1t
D	3t
E	1t
F	4t

SOLUZIONE

Eserizio 1

Il codice C corrispondente all'algoritmo proposto è il seguente:

```
int radice(int n) {
  int r = 1;
  while (((r+1)*(r+1)) <= n) {
    r++;
  }
  return r;
}
```

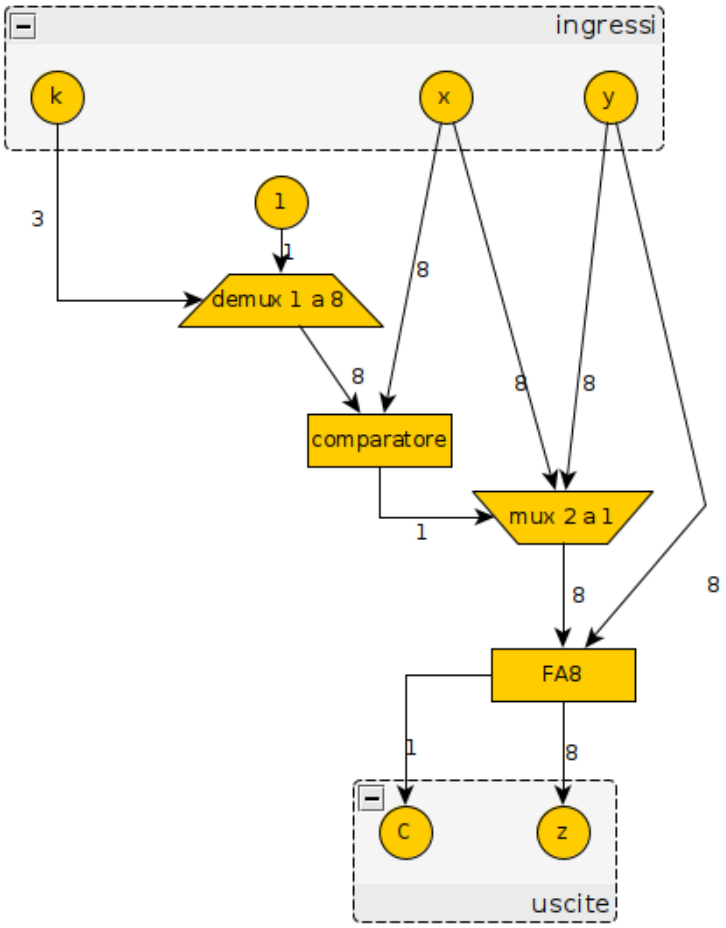
che si traduce nel codice assembler ARMv7:

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

radice:    mov r3, r0
           mov r0, #1
while:     add r1, r0, #1
           mul r2, r1, r1
           cmp r2, r3
           movgt pc, lr
           add r0, r0, #1
           b while
```

Esercizio 2

Per testare se x sia uguale a 2 alla k occorre verificare che abbia un solo 1 in posizione k. Dunque possiamo utilizzare un demultiplexer con ingresso da 1 bit fissato a 1, ingresso di controllo k da 3 bit e 8 uscite, da 1 bit ciascuna. Ricordando la forma particolare della tabella di verità del demultiplexer (ogni colonna delle uscite ha un solo 1 in corrispondenza della configurazione di ingresso di controllo che la identifica e del valore 1 dell'ingresso) il demultiplexer avrà un unico livello di porte logiche che implementato l'unico termine AND che genera l'unico 1 dell'uscita. L'uscita del demultiplexer e l'ingresso x verranno confrontati da un comparatore. Il comparatore avrà otto XOR (implementati come xy+not(x)not(y)) le cui uscite saranno poste in AND (3 livelli di logica). L'uscita del comparatore comanda un multiplexer che sceglie fra x (ingresso di controllo 0) e y (ingresso di controllo 1). Il multiplexer (di fatto 8 multiplexer con due ingressi da 1 bit e un bit di controllo, ciascuno dei quali sceglie fra il bit i-esimo di x e y) ha due livelli di logica. Infine, l'uscita del multiplexer (o x o y) andrà sommata al valore di y. Il FA da 1 bit si implementa con 2 livelli di logica. Ne servono 8 in cascata e quindi il numero dei livelli di logica sarà pari a 16. Sommando i livelli di logica componenti dei componenti che lavorano in sequenza (demultiplxer, comparatore, multiplexer, addizionatore) otteniamo dunque 22 livelli di logica, il che comporta un ritardo di stabilizzazione pari a 22 Δt.



Esercizio 3

2) Supponiamo che il tipo vettore_t contenga almeno i seguenti campi:

```
typedef struct _vettore_t {
    pthread_mutex_t mutex;      // variabile di mutua esclusione per operare sul vettore
    pthread_cond_t  cond;       // variabile di condizione per sospensione su vettore vuoto
    lista_msg_t V;              // lista di messaggi eventualmente implementata con un array
    ….
} vettore_t;
```

Supponiamo inoltre che le variabile mutex e cond siano state opportunamente inizializzate.

```
void send(vettore_t v, msg_t msg)  {
  pthread_mutex_lock(&v.mutex);
  push_back(v, msg);
  pthread_cond_signal(&v.cond);
  pthread_mutex_unlock(&v.mutex);
}
```

```
void receive(vettore_t v, msg_t *msg) {
  pthread_mutex_lock(&v.mutex);
  while(length(v)==0) {
     pthread_cond_wait(&v.cond, &v.mutex);
  }
  pop_front(v, msg);
  pthread_mutex_unlock(&v.mutex);
}
```

Per il caso vettore di capacità finita k. Si ha:

```
void send(vettore_t v, msg_t msg)  {
  pthread_mutex_lock(&v.mutex);
  while (length(v) == k) {
     pthread_cond_wait(&v.cond,  &v.mutex);
  }
  push_back(v, msg);
  pthread_cond_broadcast(&v.cond);
  pthread_mutex_unlock(&v.mutex);
}
```

```
void receive(vettore_t v, msg_t *msg) {
  pthread_mutex_lock(&v.mutex);
  while(length(v)==0) {
     pthread_cond_wait(&v.cond, &v.mutex);
  }
  pop_front(v, msg);
  pthread_cond_broadcast(&v.cond);
  pthread_mutex_unlock(&v.mutex);
}
```

Esercizio 4

Il primo processo in esecuzione sarà B, che termina l'esecuzione nel quanto di tempo che gli viene assegnato. Quindi va in esecuzione C, che termina anch'esso. A questo punto il processo col tempo residuo minore sarebbe E. Però abbiamo già fatto due assegnazioni del qdt SJF e il processo A ha un tempo residuo di 2t. Va quindi in esecuzione A e al termine del quanto di tempo, risulta avere ancora un tempo residuo di un 1 quanto di tempo. Ora è di nuovo un turno SJF. Nell'ordine FIFO, A viene prima di E, quindi va ancora in esecuzione e termina. Segue l'esecuzione di E, che termina, e quindi quella di D, che termina un quanto di tempo con un residuo di 2t. A questo punto va ancora in esecuzione D, fino al completamento, visto che F necessita di 4t. L'esecuzione termina con l'esecuzione di F per 4 qdt.

Processo	**Parte**	**Termina**	**Tempo di permanenza**
A	2	4	4
B	0	1	1
C	1	2	2
D	5	8	8
E	4	5	5
F	8	12	12

Il tempo medio di permanenza nel sistema dei processi è pari a $32t/6 = 5.33t$.