Sistemi di Calcolo (A.A. 2023-2024)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

A

Compito (23/01/2025) – Durata 1h 30'

Inserire nome, cognome e matricola nel file `studente.txt`.

ISTRUZIONI PER STUDENTI DSA: svolgere a scelta due parti su tre.

Parte 1 (programmazione IA32)

In questo esercizio si chiede di tradurre in assembly una funzione che applica una trasformazione ad un array di input, scrivendo il risultato in un array di output. In particolare, la trasformazione consiste nel sostituire l'elemento corrente con la media dei valori precedenti solo se la somma dei valori precedenti è superiore al valore dell'elemento stesso. In caso contrario, l'elemento rimane invariato. Nella directory `E1`, si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `e1A.s`:

```
unsigned transform(unsigned* array, unsigned len, unsigned* out)
{
    int i;
    unsigned sum = 0;

    for (i = 0; i < len; i++) {
        if (sum > array[i]) {
            out[i] = average(sum, i + 1);
        }
        else {
            out[i] = array[i];
        }
        sum += array[i];
    }

    return sum;
}
```

L'unico criterio di valutazione è la correttezza. Generare un file eseguibile `e1A` con `gcc -m32 -g`. Per i test, compilare il programma insieme al programma di prova `e1A_main.c`.

Non modificare in alcun modo `e1A_main.c`. Prima di tradurre il programma in IA32 si suggerisce di scrivere nel file `e1A_eq.c` una versione C equivalente più vicina all'assembly.

Parte 2 (programmazione di sistema POSIX)

Si consideri un software per la prenotazione di appuntamenti in uno studio medico. L'elenco degli appuntamenti è salvato su un file testuale i cui record hanno la seguente struttura:

Descrizione	Cognome	Giorno	Orario	Durata (minuti)
Dimensione	30 bytes	10 bytes	5 bytes	3 bytes

Tutti i campi sono rappresentati da stringhe senza terminatore. Non esiste separatore tra i campi

e tra i record. I bytes in eccesso sono costituiti da padding rappresentato con il carattere '_'. Il campo Giorno ha il formato "gg/mm/aaaa" mentre il campo Orario ha il formato "hh:mm". I record, uno per riga, sono ordinati in ordine di data ed orario crescenti.

Si scriva in e2A.c una funzione getFirstEmptySlot con il seguente prototipo:

```
int getFirstEmptySlot(const char * filename,
                      int len, char ** output);
```

che, dato in ingresso il nome filename del file contenente l'elenco delle prenotazioni e una durata espressa in minuti len, restituisce in output l'orario del primo appuntamento disponibile di durata almeno pari a len. L'orario dovrà essere rappresentato da una stringa con il formato "gg/mm/aaaa hh:mm". È importante che il codice tenga in considerazione quanto segue:

1. Le prenotazioni possono essere inserite solo tra il 18/09/2023 ed il 22/09/2023 (inclusi).
2. Le prenotazioni inserite non possono avere orario di inizio precedente le 08:00 e orario di fine successivo alle 18:00 di ogni giorno.
3. Le prenotazioni non possono sovrapporsi come orario (una prenotazione non può terminare dopo l'inizio della successiva)
4. Ogni prenotazione deve essere limitata ad un singolo giorno (non può iniziare un giorno e terminare il giorno dopo)
5. Se una richiesta non può essere soddisfatta per mancanza di un orario di appuntamento che soddisfi le condizioni precedenti la funzione deve restituire il valore -1. Lo stesso valore deve essere restituito in caso di errore nell'accesso a filename. In tutti gli altri casi dovrà essere restituito il valore 0.

Esempio: dato il file filename con il seguente contenuto:

```
Rossi_____________________________18/09/202314:00240
Marrone___________________________19/09/202308:00240
Viola_____________________________21/09/202309:00480
```

il primo appuntamento disponibile di durata pari a 420 minuti (7 ore) risulta essere "20/09/2023 08:00".

Suggerimento #1: Si ricorda che per convertire una stringa in un numero intero è possibile utilizzare la funzione atoi.

Suggerimento #2: Per rendere più semplice l'implementazione della funzione getFirstEmptySlot vengono fornite (ossia sono già implementate) due funzioni ausiliarie:

- La funzione:

  ```
  int calculateTimeDifference(const char* time1,
                              const char* time2);
  ```

  calcola quanti minuti intercorrono tra time1 e time2 espressi come "hh:mm".

- La funzione:

  ```
  char* addMinutesToTime(const char* inputTime,
                         int minutesToAdd);
  ```

  calcola un nuovo orario espresso come "hh:mm" aggiungendo minutesToAdd minuti a inputTime (espresso come "hh:mm"). Il puntatore ritornato punta ad un buffer allocato con malloc.

Per i test, compilare il programma insieme al programma di prova e2A_main.c fornito, che

non deve essere modificato. **Nota**: non modificare il file `booked1.txt e booked2.txt` che riportano un esempio di file contenente alcune prenotazioni.

Parte 3 (quiz)

Si risponda ai seguenti quiz, inserendo le risposte (A, B, C, D o E per ogni domanda) nel file `e3A.txt`. Una sola risposta è quella giusta. Rispondere E equivale a non rispondere (0 punti).

Domanda 1 (cache)

Si consideri una cache associativa a 2 vie con 4 linee da 32 byte ciascuna e politica di rimpiazzo LRU, inizialmente vuota. I blocchi dispari sono mappati sulle prime due linee, ed i blocchi pari sulle seconde due. Potendo scegliere fra più linee vuote, si usa la linea con indice più basso. Si ha inoltre un processo che accede in sequenza ai seguenti indirizzi di memoria (senza interruzioni): `389, 152, 759, 413, 820, 1273, 2420`.

Alla fine della sequenza di accessi, quali sono gli indici dei blocchi contenuti nelle 4 linee di cache? Il trattino indica che la linea di cache rimane vuota.

A	23, 39, -, -	**B**	39, 25, 12, -
C	39, 75, 12, 4	**D**	12, 4, 23, 25

Motivare la risposta nel file `M1.txt`. **Risposte non motivate saranno considerate nulle**.

Domanda 2 (pipelining)

Si consideri la seguente sequenza di istruzioni:

```
movl $5, %eax
addl %esi, %edx
subl $10, %ecx
incl %edx
movl $3, %edi
```

Riordinando le istruzioni (senza cambiare la semantica), è possibile ottimizzare il codice in modo da ridurre il numero di cicli di clock richiesti?

A	No, non è possibile	**B**	Si, è possibile ridurre il numero di cicli di clock a 9, ma non a meno
C	Si, è possibile ridurre il numero di ciclo di clock a 10, ma non a meno	**D**	Si, è possibile ridurre il numero di cicli di clock, ma le risposte B e C non sono corrette

Motivare la risposta nel file `M2.txt`. **Risposte non motivate saranno considerate nulle**.

Domanda 3 (Analisi delle prestazioni del software)

Di quanto è necessario ridurre una porzione di un programma che richiede il 75% del tempo di esecuzione per ottenere uno speedup sul programma di ~1.42?

A	20%	**B**	30%
C	40%	**D**	50%

Motivare la risposta nel file `M3.txt`. **Risposte non motivate saranno considerate nulle**.

Domanda 4 (Allineamento in memoria)

Si consideri la seguente `struct` C:

```
typedef struct S {
  char x;
  char *y;
  short z;
} S;
```

Quale delle seguenti affermazioni è **falsa**?

A	`sizeof(S) = 12`	**B**	Sono necessari 5 byte di padding
C	Il campo `z` si trova a offset `12`	**D**	Cambiando l'ordine dei campi, è possibile ottimizzare la `sizeof` di `S`

Motivare la risposta nel file `M4.txt`. **Risposte non motivate saranno considerate nulle**.