Sistemi di Calcolo (A.A. 2024-2025)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

B

Compito (19/06/2025) – Durata 1h 30'

Inserire nome, cognome e matricola nel file `studente.txt`. Se il file non viene correttamente compilato, il compito non sarà valutato.

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

Parte 1 (programmazione IA32)

La funzione `check` verifica la correttezza di una sequenza di mosse nel gioco degli scacchi. Le mosse sono memorizzate in una lista `head`, e la funzione restituisce 1 se e solo se le mosse sono tutte valide, altrimenti restituisce 0. Ogni mossa viene controllata indipendentemente dalle altre, quindi un pezzo che alla fine della mossa precedente aveva delle coordinate, nella mossa successiva potrebbe essere in una posizione completamente differente. Il controllo della singola mossa viene demandato alla funzione `is_valid_move`, la quale non tiene conto di mosse speciali, come l'arrocco, e assume che il pedone si muova sempre di una posizione. Ogni mossa è descritta in una struttura omonima (che potete trovare nel file `e1B.h`) contenente il nome del pezzo (in italiano), le coordinate iniziali, le coordinate finali e un puntatore alla mossa successiva.

In questo esercizio, si chiede di tradurre in assembly la funzione `check` scrivendo il relativo codice nel file `e1B.s` presente nella directory `E1`.

```
unsigned char check(mossa* head) {
    unsigned char count = 0;
    unsigned char correct = 0;


    while (head) {
        count++;
        if (is_valid_move(head))
            correct++;
        head = head->next;
    }
    return (count == correct)? 1 : 0;
}
```

L'unico criterio di valutazione è la correttezza. Generare un file eseguibile `e1B` con `gcc -m32 -g`. Per i test, compilare il programma insieme al programma di prova `e1B_main.c`.

Non modificare in alcun modo `e1B_main.c.` e `e1B.h`. Prima di tradurre il programma in IA32 si suggerisce di scrivere nel file `e1B_eq.c` una versione C equivalente più vicina all'assembly. Il contenuto del file `e1B_eq.c` NON sarà oggetto di valutazione.

Parte 2 (programmazione di sistema POSIX)

Si scriva una funzione che processi parallelamente un file di testo suddiviso in paragrafi. Nello specifico, si scriva nel file `E2/e2B.c` una funzione con il seguente prototipo (definito in `E2/e2B.h`):

```
void frasePiuLungaPerParagrafo(const char* nomefile, int*
numeroParagrafi, int** lunghezzaFraseMaxPerParagrafo)
```

che, dato in input il nome `filename` del file contenente del testo, restituisca alle locazioni di memoria puntate da `numeroParagrafi` e `lunghezzaFraseMaxPerParagrafo` rispettivamente il numero di paragrafi presenti nel file e una lista di interi che rappresenta la lunghezza della frase più lunga presente in ogni paragrafo, nello stesos ordine con cui i paragrafi sono presenti in `filename`. L'analisi dei paragrafi dovrà avvenire **in modo parallelo**, ossia in modo tale che ogni paragrafo venga elaborato da un un **thread differente**. Paragrafi distinti sono separati da una linea vuota. Ciascuna frase in un paragrafo termina con il carattere '`.`'.

Per esempio, se `filename` contenesse:

> *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.*
>
> *Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

Avremmo 2 paragrafi che dovranno essere analizzati da 2 thread distinti. Il primo thread conterà 2 frasi di 123 e 107 caratteri rispettivamente contenute nel primo paragrafo, mentre il secondo thread conterà una sola frase di 110 caratteri contenuta nel secondo paragrafo. La funzione, dopo aver atteso il completamento dei thread, restituirà quindi i seguenti valori nelle rispettive variabili:

- `numeroParagrafi = 2`
- `lunghezzaFraseMaxPerParagrafo = { 123, 110 }`

Si noti che il punto (carattere '`.`') che termina una frase deve essere contato come parte della sua lunghezza. Si ricorda inoltre che il carattere di ritorno a capo è rappresentato da "`\n`". Si assuma che il numero massimo di paragrafi in `filename` sia 1024, che la lunghezza massima di un paragrafo sia 10000 byte e che una riga non sia più lunga di 1024 byte (si possono utilizzare le macro definite in `e2B.h`.) Si ricorda infine di utilizzare il flag di compilazione `-lpthread` per poter utilizzare i thread POSIX.

Per i test, compilare il programma insieme al programma di prova `e2B_main.c` fornito, che **non** deve essere modificato. Inoltre, **non modificare per nessun motivo** i file `lipsum*.txt` che riportano esempi di file.

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 (assembly)

Assumendo che il registro EAX contenga il valore 0xFFFFFFFA, quale delle seguenti istruzioni produce un risultato diverso da tutte le altre?

A	`shll $3, %eax`	**B**	`leal (%eax, %eax), %eax`
C	`imull $8, %eax`	**D**	`leal (, %eax, 8), %eax`

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

Domanda 2 (permessi file)

Il comando `chmod` ha alterato i permessi del file `test` nel modo seguente:

```
PRIMA: rw-r--r--  1 utente gruppo  0  19 giu 10:00 test
DOPO:  rwxr-xr--  1 utente gruppo  0  19 giu 10:01 test
```

Quale dei seguenti comandi NON può aver causato l'alterazione descritta?

A	`chmod 754 test`	**B**	`chmod u=rwx,g=rx,o=r test`
C	`chmod u+x,g+x test`	**D**	`chmod 644 test`

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

Domanda 3 (memoria)

Si consideri un semplice allocatore di memoria dove, potendo scegliere, la `malloc` riusa il blocco libero con l'indirizzo più basso. Assumere per semplicità che non vi sia alcun header e che i blocchi allocati vengono arrotondati a una dimensione multipla di 4 byte. Inoltre, l'allocatore non divide blocchi liberi in più blocchi di dimensione inferiore, né fonde eventuali blocchi liberi adiacenti in un blocco di dimensioni superiori. Qual è il contenuto dell'heap alla fine della seguente sequenza di operazioni? ("X" denota 4 byte allocati e "." denota 4 byte liberi)

```
p1 = malloc(2);
p2 = malloc(4);
p3 = malloc(16);
free(p2);
p4 = malloc(1);
free(p3);
p5 = malloc(30);
p6 = malloc(13);
free(p1);
p7 = malloc(2);
free(p4);
```

A	`|.|X|XXXX|XXXXXXXX|`	**B**	`|X|.|....|XXXXXXXX|`
C	`|X|.|XXXX|XXXXXXXX|`	**D**	`Nessuna delle precedenti`

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

Domanda 4 (pipeline)

Si consideri la seguente sequenza di istruzioni che vengono eseguite da una CPU con pipeline a 5 stadi (`Fetch, Decode, Execute, Memory, Write-Back`)

```
0: addl $0x01, %edx
1: movl (%esp),%ebx
2: addl $0x10, %eax
3: movl $0x41, %ecx
4: addl $0x42, %edx
```

Quale tipo di hazard si verifica al momento dell'esecuzione dell'istruzione nella linea 4?

A	`Hazard strutturale`	**B**	`Hazard sui dati`
C	`Hazard sul controllo`	**D**	`Non si verifica nessun hazard`

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