**Sistemi di Calcolo (A.A. 2015-2016)**
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

B

**Esame del 20/1/2016 (esonerati) – Durata 1h 30'**

---

**Esercizio 1 (gerarchie di memoria)**

Si consideri la seguente funzione che opera su una matrice `n` x `n` di interi:

```
int matrix_sum1(int** v, int n) {
    int i, j, sum = 0;
    for (j=0; j<n; j++)
        for (i=0; i<n; i++) sum += v[i][j];
    return sum;
}

int matrix_sum2(int** v, int n) {
    int i, j, sum = 0;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) sum += v[i][j];
    return sum;
}
```

Assumere che: 1) le variabili del programma siano tutte tenute in registri della CPU; 2) le funzioni siano eseguite su un sistema con una cache da 16 KB suddivisa in linee da 32 byte; 3) tutte le righe della matrice siano allineate ad indirizzi multipli di 32; 4) `n` = 1000; 4) `sizeof(int)==4`.

1. Quanti accessi a memoria vengono effettuati nei cicli da ciascuna versione?
2. Quanti cache miss vengono generati dai cicli nello scenario peggiore nelle due versioni? Illustrare il procedimento usato per il calcolo.
3. La dimensione della linea conta? E quella della cache?

Inserire le risposte nel file `es1.txt`.

---

**Esercizio 2 (allocazione dinamica della memoria)**

Si consideri il seguente frammento di programma C eseguito su una piattaforma a 64 bit:

```
typedef struct {
    char* nome;
    int   eta;
} staff;
int i, n = 4;
staff** v = malloc(n*sizeof(staff*));
for (i=0; i<n; i++)  v[i] = malloc(sizeof(staff));
for (i=0; i<n; i+=2) free(v[i]);
for (i=0; i<n; i+=4) v[i] = malloc(sizeof(staff));
```

Si assuma che l'allocatore parta da un heap inizialmente di dimensione 4 KB. L'allocatore cercherà di minimizzare la dimensione dell'heap tentando di usare lo spazio libero con gli indirizzi più bassi. Se non si riesce a soddisfare una richiesta di allocazione, l'heap verrà espanso del minimo indispensabile. Rispondere alle seguenti domande:

1. Come è partizionato l'heap in blocchi liberi/in uso dopo ogni `malloc`/`free`?
2. Si genera frammentazione durante l'allocazione? Se sì, di che tipo?
3. Quanto è grande l'heap alla fine?

Inserire le risposte nel file `es2.txt`.

**Esercizio 3 (memoria virtuale)**

1. Supponiamo di avere uno spazio logico a 64 bit suddiviso in pagine di 8 KB. Quanti byte occuperebbe una singola tabella delle pagine che dovesse indicizzare uno spazio fisico di 16 GB? Per motivi di allineamento, assumere che le entry della tabella delle pagine siano di dimensione multipla di 32 bit.
2. Un hacker in erba legge sul manuale delle giovani marmotte una tecnica di attacco per prendere il controllo del computer della scuola su cui è installato Linux. Il suo obiettivo è alterare il comportamento del processo server Web, permanentemente attivo nel sistema, che permette di accedere voti degli studenti. Ha preso 4 in informatica e non vuole che i genitori lo scoprano. Per fare questo, deve sovrascrivere i 7 byte iniziali della funzione `read_db` del processo server in esecuzione. Durante la ricreazione, entra come utente in un computer del laboratorio e scrive un programma C di cui riportiamo un frammento:

```
void crack() {
    void* addr_to_crack =
            p1r4t4_1nd1r1zz0("/usr/sbin/apache2", "read_db");
    char patch[] = { 0x02, 0x24, 0x35, 0x01, 0x00, 0x58, 0x37 };
    memcpy(addr_to_crack, patch, sizeof(patch));
}
```

L'hacker richiama la funzione `p1r4t4_1nd1r1zz0`, che ha copiato da `spockoverflow`, che restituisce l'indirizzo del primo byte della funzione `read_db` nello spazio virtuale del processo server Web attivo nel sistema. Cosa vi aspettate che succeda quando viene eseguita la funzione `crack`[1]? Perché?
3. Il seguente testo contiene errori: "Per tradurre un indirizzo logico in un indirizzo fisico in un sistema di memoria virtuale con pagine da 8 KB, il MMU utilizza come offset i 10 bit più significativi dell'indirizzo logico e come numero di pagina i rimanenti 22". Correggere gli errori scrivendo il testo corretto.

Inserire le risposte nel file `es3.txt`.

**Esercizio 4 (ottimizzazione di programmi)**

Si crei nel file `es4-opt.c` una versione ottimizzata del seguente modulo `es4.c`:

```
#include "es4.h"

// calcola le somme degli elementi di "in" a distanza "stride"
// fra loro a partire da "from" fino a "to"
static int sum_stride(const int* in, int from, int to,
                      int* out, int stride) {
    int i, sum = 0;
    for (i=from; i<to; i+=stride) sum += in[i];
    return sum;
}

// calcola in "out[i]" le somme degli elementi di "in" a distanza
// "stride" fra loro a partire da "i"
void stat(const int* in, int n, int* out, int stride) {
    int i;
    for (i=0; i<stride; i++)
        out[i] = sum_stride(in, i, n, out, stride);
}
```

Compilare due versioni del programma, usando `gcc` **a 32 bit** con livello di ottimizzazione 1 e

[1] In quel momento, entra nel laboratorio il prof. di informatica che lo incoraggia a iscriversi a Ingegneria Informatica e Automatica alla Sapienza per iniziare a fare sul serio ☺.

lo stesso modulo `es4-main.c`:

1. Non ottimizzata: eseguibile `es4`.
2. Ottimizzata: eseguibile `es4-opt`.

Usare `gprof` per identificare le porzioni più onerose computazionalmente nelle due versioni. Chiamare gli eseguibili usati per la profilazione `es4-pg` e `es4-opt-pg`. Salvare i report di `gprof` nei file `es4.txt` ed `es4-opt.txt`, rispettivamente.

Rispondere alle seguenti domande:

1. Descrivere le ottimizzazioni applicate e dire perché si ritiene che siano efficaci. Cercare di capire se c'è qualche risorsa che il programma usa male (suggerimento: cache). Usare il buon senso!
2. Riportare il tempo di esecuzione di `es4` e di `es4-opt` usando il comando `time`.
3. Riportare i flat profile delle due versioni usando `gprof`.
4. Di quante volte è più veloce l'eseguibile `es4-opt` rispetto a `es4`?
5. Usando i dati del profilo `es4.txt`, calcolare lo speedup che bisognerebbe ottenere per la funzione `stat` (considerandone il tempo complessivo self+children) per ottenere uno speedup totale per l'intero programma pari a 2x, se questo è possibile. Motivare la risposta.

Inserire le risposte nel file `es4.txt`. Alla fine del compito, **non eliminare i seguenti file**:

- `es4`
- `es4-pg`
- `es4.txt`
- `es4-opt`
- `es4-opt-pg`
- `es4-opt.txt`
- `gmon.out`