Sistemi di Calcolo (A.A. 2016-2017)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

A

Esame del 27/1/2017 (esonerati) – Durata 1h 30'

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

Parte 1 (allocazione dinamica della memoria)

Dato il seguente frammento di codice:

```
typedef struct pair {
    short   info;
    pair    *next;
} pair;

pair* init(short s, int n) {
    pair* v = calloc(n, sizeof(pair));
    while (size--) v[size].info = s;
    return v;
}

pair *a, *b, *c, *d;
a = init(4, 2);
b = init(7, 3);
c = init(3, 2);
free(a);
a = init(2, 1);
free(b);
b = init(9, 5);
d = init(3, 2);
```

Si assuma che l'allocatore parta da un heap inizialmente vuoto. Esso 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, motivando le risposte:

1. Quanti byte sono richiesti per rappresentare una `struct pair` in memoria?
2. Come è partizionato l'heap in blocchi liberi/in uso dopo ogni `init` /`free`?
3. Si genera frammentazione durante l'esecuzione del programma? Se sì, di che tipo?
4. Quanto è grande l'heap alla fine?
5. Cambiando l'ordine delle prime tre `init` si può ridurre la dimensione finale dell'heap?

Si assuma una architettura a 32 bit. Inserire le risposte nel file `es1A.txt`.

Parte 2 (programmazione syscalls)

Si completi il generatore di numeri primi `primer.c` di modo che:

1. all'avvio del programma venga creato un file di output denominato `out.txt` (se il file esiste già, troncarne il contenuto) con permessi `0644`;
2. `writeToFile()` scriva i primi `len` bytes nel buffer `buf` sul descrittore di file `fd`;
3. alla ricezione di un segnale `TERM` o alla pressione di CTRL-C da parte dell'utente la variabile globale `shouldStop` venga settata ad `1` (senza uscire immediatamente);
4. in uscita dal ciclo `while` del metodo `main` il descrittore del file venga chiuso.

Gestire eventuali errori restituiti dalle system call invocate. Per semplicità al punto 2 si assuma che interruzioni e scritture parziali non possano verificarsi. Compilare usando `make`.

Parte 3 (analisi e ottimizzazione località)

Sia dato il seguente frammento di codice:

```
int mcopy(int *dest, int *src, int n, int m) {
    if (n%m) return -1; // n deve essere multiplo di m
    int i, j;
    for (i=0; i<m; i++) {
        int index = m-i-1;
        for (j=i; j<n; j+=m)
            dest[j] = src[index];
    }
    return 0;
}
```

1. Si assuma di eseguire `mcopy` in un sistema di calcolo con un livello di cache e potendo usare solo due linee da 32 byte ciascuna. Si assuma inoltre che tutte le variabili siano mantenute nei registri, che `dest` e `src` siano allineati a 32 byte, che `n=2048` e che `m=8`.

   a. Quanti accessi a memoria vengono effettuati?
   b. Determinare il numero di hit e miss per la cache
   c. Discutere il ruolo della dimensione delle linee di cache
   d. Come cambierebbe il numero di hit e miss se invece di avere a disposizione soltanto due linee, si potesse utilizzare l'intera cache (di capacità 16 KB)?

Inserire le risposte all'interno del file `es3A.txt`.

2. Implementare nel file `mcopy-opt.c` una versione ottimizzata del codice riportato sopra. Tramite `make` sarà possibile compilare due programmi di prova: `mcopy` per eseguire il codice riportato in alto e `mcopy-opt` per quello ottimizzato. Non è richiesto riportare lo speedup.

Parte 4 (quiz)

Si risponda ai seguenti quiz, inserendo le risposte (A, B, C, D o E per ogni domanda) nel file `es4A.txt`. **Una sola risposta è quella giusta**. Rispondere E equivale a non rispondere (0 punti). Le motivazioni vanno inserite in un file diverso per ciascuna domanda (si veda sotto).

Domanda 1 (endianness)

Si assuma di operare in una architettura IA32 sul seguente frammento di memoria:

Indirizzo	0x1000	0x1001	0x1002	0x1003
Contenuto	0xA1	0xB2	0xC3	0xD4

Eseguendo le seguenti istruzioni:

```
movw $0xF5E6, 0x1002
movl 0x1000, %eax
```

Cosa conterrà il registro `%eax`?

A	0xF5E6A1B2	**B**	0xF5E6B2A1
C	0xA1B2F5E6	**D**	0xE6F5B2A1

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

Domanda 2 (paginazione)

Quali dei seguenti componenti non è essenziale in una architettura con memoria virtuale?

A	Cache	**B**	Disco rigido
C	MMU	**D**	Exception handler

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

Domanda 3 (flusso di controllo eccezionale)

Una delle seguenti affermazioni è **falsa**, quale?

A	L'Interrupt Descriptor Table (IDT) è una peculiarità delle architetture x86/x86-64 per l'implementazione di un interrupt vector	**B**	Una trap originata con una istruzione INT prevede che soltanto i registri callee-save vengano salvati automaticamente prima di eseguire l'interrupt handler
C	Un interrupt vector associa una lista di puntatori ad interrupt handler ai codici di interruzione corrispondenti	**D**	L'esecuzione di un interrupt handler può richiedere che vengano eseguite istruzioni privilegiate della CPU

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

Domanda 4 (architetture di calcolo)

Su una architettura con pipeline, quale delle seguenti affermazioni è **falsa**?

A	Un valore alto di CPI può suggerire la presenza di "bolle" nella pipeline	**B**	Riordinare una sequenza di istruzioni può migliorare l'instruction-level parallelism che si osserva nell'esecuzione
C	La scelta di un compilatore di privilegiare istruzioni che occupano meno byte (es. `incl` vs. `addl $1`) dà talvolta luogo a "bolle" nella pipeline	**D**	Un errore nel processo di branch prediction di norma comporterà un flush (svuotamento) della pipeline

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