Sistemi di Calcolo (A.A. 2017-2018)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

C

Compito di esonero (16/01/2018) – Durata 1h 30'

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

Parte 1 (programmazione IA32)

Nella directory `es1C`, si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `es1C.s`:

```
int diff(int x, char y);

int check(int* a, char* b, unsigned n) {
    int* pa = a + n - 1; // occhio all'aritmetica dei puntatori!
    while (pa >= a) {
        int res = diff(*pa--, *b++);
        if (res & 8 || res & 128 || res & 2048) return 1;
    }
    return 0;
}
```

L'unico criterio di valutazione è la correttezza, cioè l'equivalenza semantica tra il programma tradotto e quello C di partenza. Generare un file eseguibile `es1C` con `gcc -m32 -g`. Per i test, compilare il programma insieme al programma di prova `es1C-main.c` e al modulo `prod.s`.

Nota: **non** modificare in alcun modo `es1C-main.c` e `diff.s`.

Parte 2 (programmazione di sistema POSIX)

Si scriva un programma `execn` in ambiente POSIX che, dati come argomenti il nome di un programma eseguibile `cmd` ed un intero `n`, lancia `n` processi figli: l'`i`-esimo processo lanciato deve eseguire il programma `cmd` passandogli come argomento il numero `i`. Dopo aver lanciato tutti i processi figli, `execn` deve attendere la loro terminazione: se almeno uno fra i processi figli termina con uno stato di terminazione diverso da zero, allora `execn` deve terminare con uno stato di terminazione diverso da zero, altrimenti `execn` deve terminare con stato di terminazione di successo.

Usare `execvp` per l'esecuzione in modo che il programma eseguibile venga cercato in tutte le directory della variabile ambiente `PATH`.

Per convertire una stringa in un intero utilizzare la funzione `atoi`. Per convertire un intero in una stringa usare la funzione `sprint`, dove `sprintf(buf, "%d", k)` converte l'intero `k` in una stringa, memorizzandola nel buffer `buf`.

Sintassi: `execn cmd n`

Gestione degli errori: `execn` deve terminare con stato di terminazione diverso da zero se una qualsiasi delle chiamata POSIX fallisce.

Esempio:

```
> ./execn a
usage: ./execn <cmd> <n>
> echo $?
1
> ./execn a 5
```

```
errore nella exec: No such file or directory
errore nella exec: No such file or directory
errore nella exec: No such file or directory
errore nella exec: No such file or directory
errore nella exec: No such file or directory
> echo $?
1
> ./execn sleep 5
> echo $?
0
```

Si noti come `echo $?` stampi il codice di terminazione del comando precedente.

Parte 3 (ottimizzazione)

Sia dato il seguente frammento di codice:

```
typedef struct node_t {
    unsigned elem[2];
    struct node_t* next;
} node_t;

unsigned get(const node_t* node, int i) { return node->elem[i]; }

unsigned sum_i(const node_t* list, int i) {
    const node_t* p;
    unsigned sum = 0;
    for (p=list; p != NULL; p = p->next) sum += get(p, i);
    return sum;
}

unsigned sum(const node_t* list) {
    return sum_i(list, 0) + sum_i(list, 1);
}
```

Compilare due versioni del programma, usando `gcc` a 32 bit con livello di ottimizzazione 1 e lo stesso modulo `es3C-main.c`:

1. **non** ottimizzata manualmente: eseguibile `es3C`;
2. ottimizzata **manualmente**: eseguibile `es3C-opt`.

Ai fini dell'ottimizzazione:

1. usare `gprof` per identificare le porzioni più onerose computazionalmente. Per evitare confusione, chiamare l'eseguibile usato per la profilazione `es3C-pg` e il report del profiler `es3C-pg.txt`;
2. esaminare il modulo `es3C.s` generato a partire da `es3C.c` con `gcc -m32 –S -O1` (già fornito) per capire quali ottimizzazioni siano **già** state effettuate dal compilatore.

Rispondere alle seguenti domande:

1. Descrivere le ottimizzazioni applicate **manualmente** e dire perché si ritiene che siano efficaci.
2. Riportare i tempi di esecuzione (real) di **tre** run di `es3C` e di `es3C-opt` e la loro media. I tempi possono essere ottenuti usando il comando time.
3. Riportare lo speedup di `es3C-opt` rispetto `es3C`

Inserire le risposte nel file `es3C.txt`. Alla fine del compito, **non** cancellare `gmon.out` e gli altri eseguibili creati.

Parte 4 (quiz)

Si risponda ai seguenti quiz, inserendo le risposte (A, B, C, D o E per ogni domanda) nel file `es4C.txt`. Una sola risposta è quella giusta. Rispondere E equivale a non rispondere (0 punti). Risposte errate valgono 0 punti. Le motivazioni vanno inserite nel file `motivazioni.txt`. **Risposte non motivate saranno considerate nulle.**

Domanda 1 (paginazione)

Quante entry contiene la tabella delle pagine di un processo in un sistema che ha 4 GB di memoria fisica, frame da 64 KB e spazio logico pari a 2 GB?

A	32768	**B**	4096
C	65536	**D**	16384

Domanda 2 (permessi)

Si vogliono impostare i permessi del file paperino. Quale comando andrebbe eseguito per settare permessi di sola lettura per il proprietario, permesso di sola esecuzione per il gruppo e permesso di sola scrittura per altri?

A	`chmod 0124 paperino`	**B**	`chmod 0142 paperino`
C	`chmod 0412 paperino`	**D**	`chmod 0214 paperino`

Domanda 3 (analisi prestazioni)

Qual è lo speedup ottenibile per un programma se riduciamo del 60% il tempo di esecuzione di una sua porzione A che richiede il 40% del tempo complessivo di esecuzione?

A	~1.12x	**B**	~1.22x
C	~1.32x	**D**	~1.52x

Si noti che se prima dell'ottimizzazione il tempo di una porzione è T_A, dopo l'ottimizzazione è $T_A' = T_A - 0.6 * T_A$

Domanda 4 (sistema di calcolo)

Quale delle seguenti affermazioni è **vera**?

A	un segnale è sintomo di una condizione di errore in un programma	**B**	lo spazio logico di un processo può essere più grande della memoria fisica del sistema
C	una trap porta alla generazione di un segnale	**D**	un sistema può andare in thrashing se la cache è piccola