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

B

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

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

Parte 1 (programmazione IA32)

Nella directory `es1B`, si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `es1B.s`:

```
int prod(short x, char y, int * res);

int check(short* a, char* b, unsigned n) {
    short* pa = a + n - 1; // occhio all'aritmetica dei puntatori!
    while (pa >= a) {
          int res;
          prod(*pa--, *b++, &res);
        if (res == 6) 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 `es1B` con `gcc -m32 -g`. Per i test, compilare il programma insieme al programma di prova `es1B-main.c` e al modulo `prod.s`.

Nota: **non** modificare in alcun modo `es1B-main.c` e `prod.s`.

Parte 2 (programmazione di sistema POSIX)

Si scriva un programma `waiter.c` in ambiente POSIX che, dati come argomenti il percorso di un programma eseguibile ed un suo argomento, lancia un'istanza del programma eseguibile, passandogli il relativo argomento. Prima di lanciare l'esecuzione del programma, `waiter` deve:

1. verificare che siano stati passati gli argomenti necessari (nome di un programma eseguibile ed un argomento). In caso di errore, `waiter` deve stampare un messaggio di errore sullo standard error e immediatamente terminare con uno stato di terminazione diverso da zero.
2. verificare che sia stata settata nell'ambiente la variabile `WAIT` e che il suo valore sia uno tra "`TRUE`" o "`FALSE`". In caso contrario, `waiter` deve stampare un messaggio di errore sullo standard error e immediatamente terminare con uno stato di terminazione diverso da zero.

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

Dopo aver lanciato il programma, se la variabile `WAIT` è impostata al valore `TRUE`, `waiter` deve attendere la terminazione del programma lanciato e restituirne lo stato di terminazione. Se la variabile `WAIT` è settata a `FALSE`, `waiter` deve restituire lo stato di terminazione di successo.

Sintassi: `waiter eseguibile arg`

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

Esempio: Si noti che nei comandi che seguono viene utilizzato `WAIT=VAL` prima di un comando per impostare la variabile `WAIT` al valore `VAL` per il processo associato al comando che si sta per eseguire.

```
> WAIT=PIPPO ./waiter
usage: ./waiter <cmd> <arg>
> echo $?
1
> WAIT=PIPPO ./waiter ls .
Environment variable WAIT is not properly set. Aborting.
> echo $?
1
> WAIT=TRUE ./waiter ls .
waiter    waiter.c
> echo $?
0
> WAIT=TRUE ./waiter ls ...
ls: cannot access '...': No such file or directory
> echo $?
1
> WAIT=FALSE ./waiter ls ...
ls: cannot access '...': No such file or directory
> echo $?
0
> WAIT=FALSE ./waiter ls .
waiter    waiter.c
> echo $?
0
```

Si noti inoltre come `echo $?` stampi il codice di terminazione del comando precedente. Si ricordi che per confrontare due stringhe può essere utilizzata la funzione `strcmp`.

Parte 3 (ottimizzazione)

Sia dato il seguente frammento di codice:

```
int get(char ** m, int a, int b) { return m[a][b]; }

int bazinga(char ** m, int n) {
    int i, j, k;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            int sum = 0;
            for (k = i; k < n; k++) sum += get(m, j, k);
            m[j][i] = sum;
        }
    }
}
```

Compilare due versioni del programma, usando gcc a 32 bit con livello di ottimizzazione 1 e lo stesso modulo es3B-main.c:

1. **non** ottimizzata manualmente: eseguibile es3B;
2. ottimizzata **manualmente**: eseguibile es3B-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 es3B-pg e il report del profiler es3B-pg.txt;
2. esaminare il modulo es3B.s generato a partire da es3B.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 es3B e di es3B-opt e la loro media. I tempi possono essere ottenuti usando il comando time.
3. Riportare lo speedup di es3B-opt rispetto es3B

Inserire le risposte nel file es3B.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 `es4B.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)

Se un sistema è configurato con pagine a 1MB quanti bit sono necessari per esprimere l'offset in una pagina?

A	10	**B**	15
C	20	**D**	25

Domanda 2 (permessi)

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

A	chmod 0124 pippo	**B**	chmod 0142 pippo
C	chmod 0421 pippo	**D**	chmod 0214 pippo

Domanda 3 (analisi prestazioni)

Qual è lo speedup ottenibile per un programma se riduciamo del 25% il tempo di esecuzione di una sua porzione A che richiede il 30% del tempo complessivo di esecuzione?

A	~1.08x	**B**	~1.8x
C	~1.03x	**D**	~1.55x

Si noti che se prima dell'ottimizzazione il tempo di una porzione è T_A, dopo l'ottimizzazione è $T_A' = T_A - 0.25 * T_A$

Domanda 4 (segnali)

Quale delle seguenti affermazioni è **vera**?

A	un segnale è sempre generato a causa di un interruzione hardware	**B**	se un processo termina, anche i suoi processi figli vengono terminati dal sistema operativo
C	il segnale SIGCHLD viene consegnato ad un processo quando il suo processo padre viene terminato	**D**	se un processo riceve il segnale SIGTERM può decidere di non terminare la sua esecuzione