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

A

Esame del 16/1/2018 (esonerati) – Durata 1h 30'

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

Parte 1 (allocazione dinamica della memoria)

Si consideri il seguente programma C:

```
int** alloc_diagonal_matrix(int n) {
    int **m, i;
    m = malloc(n*sizeof(int*));
    for (i=0; i<n; ++i) m[i] = calloc(i+1, sizeof(int));
    return m;
}

void make_square(int** m, int n) { // rende quadrata matrice diagon.
    int i, j;
    for (i=0; i<n-1; ++i) {
        int* temp = calloc(n, sizeof(int));
        free(m[i]);
        m[i] = temp;
    }
}

int main() {
    int** m = alloc_diagonal_matrix(3);
    make_square(m, 3);
    return 0;
}
```

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. La minima dimensione di un blocco allocato è di 8 byte: se a `malloc` viene chiesto meno, vengono comunque allocati 8 byte. Si ricordi che `calloc(n,i)` è come `malloc(n*i)`, ma in più azzera i byte allocati.

Rispondere alle seguenti domande, motivando le risposte:

1. Come è partizionato l'heap in blocchi liberi/in uso dopo ogni `malloc/free`?
2. Si genera frammentazione durante l'esecuzione del programma? Se sì, di che tipo?
3. Quanto è grande l'heap prima della fine del main?
4. Come cambieresti `alloc_diagonal_matrix` o `make_square` in modo da ridurre la dimensione dell'heap senza modificare il comportamento del programma?

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

Parte 2 (programmazione di sistema POSIX)

Si scriva un programma `pardo.c` in ambiente POSIX che, dati come argomenti il percorso di un programma eseguibile e una lista di n argomenti, lancia n istanze del programma eseguibile, ciascuna con uno degli argomenti dati. Il programma `pardo` deve attendere la terminazione di ciascun programma. Usare `execvp` per l'esecuzione in modo che il programma eseguibile venga cercato in tutte le directory della variabile ambiente `PATH`.

Sintassi: `pardo eseguibile arg1 arg2 arg3 ... argn`

Gestione degli errori: `pardo` deve terminare con stato diverso da zero se ha meno di 3 argomenti (incluso il nome `pardo` stesso), una qualsiasi chiamata POSIX fallisce o se una qualsiasi delle n esecuzioni del programma `eseguibile` fallisce.

Compilare il programma usando il comando `make`.

Esempio:

```
> ./pardo md5sum /bin/ps /bin/sleep /bin/date
659cd1348e21304b9d0632ad6844ac2c /bin/date
413a032ed34df6f02d17d6d21fe89187 /bin/sleep
facd479db88584855d8c487eaad84882 /bin/ps
> echo $?
0
> ./pardo md5sum
usage: pardo exec-file arg1 arg2 ... argn
> echo $?
1
> ./pardo prova uno
cannot exec: No such file or directory
> echo $?
1
>  ./pardo md5sum pippo
md5sum: pippo: No such file or directory
>
```

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

Parte 3 (analisi località)

Sia dato il seguente frammento di codice:

```
void pass(int* v, int n, int stride) {
    int i;
    for (i=0; i+stride<n; i+=stride)
        if (v[i] > v[i+stride]) {
            int temp = v[i];
            v[i] = v[i+stride];
            v[i+stride] = temp;
        }
}
```

Si assuma di invocare la funzione pass[1] in un sistema di calcolo con una cache completamente associativa con 512 linee da 64 byte ciascuna. Si assuma inoltre che tutte le variabili siano mantenute nei registri, che l'indirizzo in `v` sia allineato a 64 byte e che n=1024 e stride=8.

a. Quante richieste di accesso a memoria (al più) vengono effettuate dalla funzione?
b. Quanti cache miss si hanno?
c. Cambierebbe qualcosa se il numero di linee di cache fosse diverso?

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

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 nel file **motivazioni.txt**. **Risposte non motivate saranno considerate nulle.**

---

[1] Curiosità: la funzione è un ingrediente dell'algoritmo di ordinamento shellsort.

Domanda 1 (thrashing)

Si consideri un sistema di memoria virtuale con swapping. Una sola delle seguenti affermazioni è **vera**. Quale?

A	Il thrashing è un fenomeno che impatta principalmente la correttezza dei risultati ottenuti dai processi.	**B**	Un sistema di memoria virtuale con swapping può entrare in thrashing solo se i dischi sono lenti.
C	Se le pagine in uso da parte dei processi attivi non sono mai più dei frame fisici disponibili, il sistema non entra mai in thrashing.	**D**	Il thrashing può essere sempre evitato se il sistema è dotato di cache di memoria di dimensioni confrontabili con la memoria fisica.

Domanda 2 (paginazione)

Quante entry ha la tabella delle pagine di un sistema di memoria a 48 bit con pagine di 8 KB?

A	2^{35}	**B**	2^{34}
C	2^{32}	**D**	8192

Domanda 3 (stati di un processo)

Una sola delle seguenti affermazioni sugli stati di un processo è **falsa**. Quale?

A	Un processo nello stato ready non è in esecuzione solo perché la CPU è impegnata per un altro processo.	**B**	Un processo nello stato ready può essere schedulato in qualsiasi momento per l'esecuzione.
C	Un processo passa dallo stato waiting allo stato ready quando si ha la terminazione di un'operazione di I/O.	**D**	Un processo entra sempre in stato waiting dallo stato ready.

Domanda 4 (permessi)

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

A	`chmod 600 pippo`	**B**	`chmod 744 pippo`
C	`chmod 477 pippo`	**D**	`chmod 470 pippo`