Domande su memory layout, stack, heap, etc.

Domanda 1

Guarda il seguente codice:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    // Allocazione di memoria dinamica per un intero
    int *heap_var = (int*)malloc(sizeof(int));

    // Verifica che malloc abbia avuto successo
    if (heap_var == NULL) {
        fprintf(stderr, "Errore di allocazione della memoria.\n");
        return 1; // Esci se l'allocazione fallisce
    }

    // Assegna un valore all'area di memoria allocata
    *heap_var = 42;

    // Stampa il valore
    printf("Il valore allocato nell'heap è: %d\n", *heap_var);

    // Non chiamando free(heap_var), si verifica un memory leak

    return 0; // La memoria allocata non viene liberata
}
```

Nella funzione main, viene utilizzata la funzione malloc per allocare memoria dinamicamente. In quale sezione della memoria viene allocata la variabile puntata da heap_var? Cosa accade alla memoria allocata dopo l'esecuzione del programma se la funzione free non fosse stata chiamata? Il termine memory leak ti dice qualcosa?

Soluzione

- La variabile puntata da heap_var viene allocata nella sezione **heap** della memoria. L'heap è una zona dedicata alla memoria dinamica. In C, è gestita manualmente dal programmatore, utilizzando funzioni come malloc, calloc, realloc, e free.
- In C, la gestione della memoria è a carico del programmatore. Se un'area di memoria viene allocata dinamicamente ma non è più accessibile (ad esempio, se non c'è più un puntatore che la riferisce) e non viene deallocata con la funzione free, si verifica un **memory leak**. Un memory leak si verifica quando della memoria resta occupata ma non viene più utilizzata, rendendo tale memoria inutilizzabile per il programma fino al termine della sua esecuzione.

Domanda 2

Descrivi le principali differenze tra il segmento Heap e il segmento Stack.In particolare, fai riferimento a come avviene l'allocazione e la deallocazione in ciascuno di essi facendo riferimento a C o a Java, tipologia di dati che vi risiedono, come vengono allocate e deallocate le variabili in ciascuna delle due aree.

Soluzione

Stack:

Lo **stack** è una sezione di memoria che cresce in direzione opposta rispetto all'**heap**. È una struttura dati di tipo **LIFO** (Last In, First Out) utilizzata per gestire le chiamate a funzione e le variabili locali.
• **Allocazione e deallocazione**: L'allocazione nello stack avviene automaticamente quando si entra in una funzione e viene creato uno **stack frame**. Questo frame contiene:

• I parametri della funzione
• Le variabili locali
• L'indirizzo di ritorno, utilizzato per riprendere l'esecuzione del programma dopo il termine della funzione.

La deallocazione è altrettanto automatica: quando una funzione termina, lo stack frame corrispondente viene rimosso tramite un'operazione di **pop**, e il controllo del programma ritorna alla funzione chiamante.

• **Tipologia di dati**: Nello stack risiedono principalmente le **variabili locali** e i **parametri di funzione**. In linguaggi come C, le variabili automatiche (cioè le variabili locali non dichiarate come static o allocate dinamicamente) sono allocate nello stack. Anche in Java, le variabili primitive locali (come int, float, ecc.) e i riferimenti agli oggetti sono allocati nello stack, anche se gli oggetti stessi risiedono nell'heap.

• **Limiti e gestione**: Lo stack ha dimensioni limitate e predefinite, stabilite dal sistema operativo. Se il programma richiede più memoria di quanta ne sia stata assegnata allo stack (ad esempio, a causa di una ricorsione eccessiva o allocazioni troppo grandi), si verifica un errore chiamato **stack overflow**.

Heap:

L'**heap** è un segmento di memoria utilizzato per l'allocazione dinamica. A differenza dello stack, la gestione della memoria nell'heap non segue un ordine LIFO ed è completamente gestita dal programmatore (in C) o dal sistema di runtime (in Java).

• **Allocazione e deallocazione**:

• In **C**, l'allocazione della memoria nell'heap avviene tramite funzioni come malloc(), calloc(), o realloc(). La deallocazione deve essere eseguita manualmente chiamando free(). Se la memoria non viene liberata correttamente, si verifica un **memory leak**.

• In **Java**, la memoria nell'heap viene gestita automaticamente dalla **Java Virtual Machine (JVM)**. Gli oggetti creati con l'operatore new risiedono nell'heap, e la **Garbage Collection (GC)** si occupa di recuperare la memoria quando gli oggetti non sono più utilizzati.

• **Tipologia di dati**: Nell'heap risiedono i dati allocati dinamicamente, come strutture, array o oggetti. In C, ogni volta che si utilizza malloc() per creare un array o una struttura, questa viene allocata nell'heap. In Java, gli oggetti (istanze di classi) sono sempre creati nell'heap, indipendentemente da dove si trova il riferimento che li punta.

• **Limiti e gestione**: L'heap ha una gestione più flessibile rispetto allo stack, ma è più lento perché richiede l'allocazione e la deallocazione manuale (in C) o il tracciamento automatico (in Java) tramite il garbage collector. Se l'heap esaurisce la memoria disponibile (ad esempio, quando ci sono troppe allocazioni o un numero elevato di oggetti non deallocati), si verifica un errore di **OutOfMemory**.

Domanda 3

Cos'è e quando si verifica uno stack overflow? Spiega in quali situazioni può verificarsi

Risposta:
Uno **stack overflow** si verifica quando la memoria dello stack, riservata per le chiamate a funzioni e le variabili locali, viene completamente esaurita. Lo stack ha una dimensione limitata, stabilita dal sistema operativo o dall'ambiente di runtime, e quando questa viene superata, il programma non è più in grado di allocare spazio per nuove chiamate a funzioni o variabili locali, causando un errore di **stack overflow**. Si può verificare per 1) ricorsione eccessiva 2) allocazione di variabili troppo grandi (array grandi dimensioni, strutture dati complesse) 3) troppo profondità nelle chiamate a funzione

Domanda 4

Cosa significa "allocazione dinamica della memoria"? Fornisci un esempio pratico in C e in Java di come utilizzare malloc() e new.

Risposta
L'**allocazione dinamica della memoria** consente a un programma di richiedere memoria durante l'esecuzione, invece di definire la quantità di memoria necessaria in fase di compilazione. Questo approccio è utile quando la dimensione delle strutture dati non è nota in anticipo o può variare in base alle esigenze del programma.

In C, si utilizzano funzioni come malloc

```
    int *num = (int *)malloc(5 * sizeof(int));
```

In questo esempio, malloc() alloca memoria sufficiente per un numero intero. Il programmatore deve liberare la memoria usando la funzione free

```
free(num);
```

In Java, l'allocazione dinamica della memoria avviene automaticamente con l'operatore new, che crea nuovi oggetti durante l'esecuzione. La gestione della memoria in Java è automatizzata dal **Garbage Collector**, che si occupa di liberare la memoria degli oggetti non più referenziati.

```
public class Main {
    public static void main(String[] args) {
        // Allocazione dinamica di un array di 5 interi
        int[] array = new int[5];

        // Utilizzo dell'array
        for (int i = 0; i < array.length; i++) {
            array[i] = i + 1;
            System.out.print(array[i] + " ");
        }
    }
}
```

In questo esempio, new viene utilizzato per allocare un array di 5 interi. A differenza di C, non è necessario liberare manualmente la memoria, poiché la gestione viene effettuata automaticamente dalla **JVM**.

Domanda 5

Cosa significa "out of memory"? SI rifesrisce allo stack o all'heap?Spiega quali circostanze possono portare un sistema o un programma a esaurire la memoria disponibile (sia nello stack che nell'heap), e quali sono i sintomi di un problema di esaurimento della memoria.

Risposta
L'errore **"out of memory"** si verifica quando un programma o il sistema operativo non ha più memoria disponibile per allocare nuove risorse. Questo problema può interessare sia la **memoria dello heap** che la **memoria dello stack**, anche se il tipo di esaurimento e le cause possono variare a seconda del segmento.

Heap: È più probabile che l'errore "out of memory" relativo all'heap si manifesti in applicazioni che fanno un uso intensivo di allocazioni dinamiche o che soffrono di memory leak. Questo errore viene segnalato chiaramente nei moderni sistemi tramite eccezioni o messaggi di errore specifici.

Stack: L'errore nello stack è più spesso legato a un **stack overflow**, e può verificarsi quando ci sono troppi livelli di chiamate a funzioni o quando le variabili locali sono troppo grandi. A differenza dell'heap, l'esaurimento dello stack non sempre genera messaggi di errore espliciti, ma può causare un crash improvviso del programma.