Domande Memory Layout

Consegna: Rispondi alle domande in modo chiaro e dettagliato.

Domanda 1

Nel contesto della memoria di un processo, le sezioni **text**, **data**, **BSS**, **heap** e **stack** svolgono ruoli specifici. Per ciascuna di queste sezioni:

1. **Descrivi** il ruolo e la funzione della sezione all'interno del processo.

2. Indica il **posizionamento** tipico nello spazio di indirizzamento di un processo (dove si trova rispetto alle altre sezioni).

3. Spiega in che modo la sezione può **crescere** o modificarsi durante l'esecuzione del programma.

4. Descrivi come la memoria associata a quella sezione viene **gestita** o **protetta** (ad esempio, permessi di lettura, scrittura, esecuzione).

Soluzione:

-**Text**: segmento della memoria di un processo che contiene istruzioni eseguibili del programma caricato in memoria (ovvero, del processo in esecuzione). Il segmento text è posizionato sotto il segmento Data, Stack e Heap. ˜Area di memoria read-only (per impedire che le istruzioni siano modificate dal programma)

-**Data**: segmento spazio indirizzamento che contiene le variabili globali e le variabili statiche inizializzate da un programmatore. Può avere una parte a sola lettura (per esempio variabili costanti), una parte lettura/scrittura.

-**BSS**:segmento spazio indirizzamento del programma che contiene le variabili globali non esplicitamente inizializzate dal programmatore.

-**Stack**: Cresce in direzione opposta rispetto al segmento **Heap**. Questo segmento contiene lo **stack** del programma, una struttura dati **LIFO** (Last In, First Out) che tiene traccia delle chiamate a funzione. Ad ogni nuova chiamata a una funzione, viene creato uno **stack frame**, che viene poi inserito nello stack tramite un'operazione di **push**. Lo stack frame contiene i parametri formali, le variabili locali e l'indirizzo di ritorno, oltre ad altre informazioni come alcuni registri della CPU e dettagli sulla funzione chiamante. Quando la funzione termina la sua esecuzione, il controllo ritorna all'indirizzo di ritorno, e lo stack frame viene rimosso con un'operazione di **pop**. In questo modo, ogni chiamata a funzione (o metodo, ad esempio in **Java**) rimane isolata e separata dalle altre chiamate.
Quando il segmento **stack** esaurisce la memoria assegnata dal sistema operativo, si verifica un errore di **stack overflow**.

-Il **segmento heap** è utilizzato per l'allocazione dinamica della memoria, ovvero per quei dati di cui il programmatore non conosce a priori la dimensione, o che sono troppo grandi per essere allocati nello stack. È posizionato subito dopo il segmento **BSS** e cresce verso l'alto. Un utilizzo eccessivo dell'heap può causare un errore di **OutOfMemory**. In **C**, la memoria nell'heap viene gestita manualmente dal programmatore tramite chiamate di sistema come malloc(). In **Java**, gli oggetti sono creati nello heap dalla **JVM** e il **garbage collector** si occupa di liberare la memoria quando un'istanza di un oggetto non è più utilizzata.

Domanda 2

Qual è la differenza tra la sezione **Data** e la sezione **BSS**? Fornisci un esempio di variabile che potrebbe essere memorizzata in ciascuna di queste due sezioni.

Soluzione:
Nel segmento **Data** si memorizzano variabili globali inizializzate, nel segmento **BSS** non inizializzate.

```
C int var_data = 10; // inizializzata nel segmento Data static int i = 100; // statica inizializzata nel segmento Data int var_bss; // nel segmento BSS static int i; // Uninitialized static variable stored in bss int main(){ }
```

Domanda 3

Considera il seguente programma C semplificato:

```c
#include <stdio.h>
#include <stdlib.h>
int global_var = 10;
int uninitialized_global;

void func() {
    int local_var = 5;
    printf("Local var: %d\n", local_var);
}

int main() {
    int *var = (int*)malloc(sizeof(int));
    *var = 20;
    int x = 10;
    func();
    printf("var: %d\n", *var);
    free(var);
    printf("x: %d\n", x);
    return 0;
}
```

Indica in quale sezione di memoria vengono allocate le seguenti variabili:

Soluzione

1. global_var : area Data perché globale e inizializzata
2. uninitialized_global area BSS
3. local_var all'interno della funzione func() nello stack frame della funzione func perché variabile locale e visibile solo alla funzione func
4. var (il puntatore dinamico nella funzione main()) nello heap
5. x (variabile locale nella funzione main()) nello stack frame della funzione main e visibile solo al main