Bài 8: Memory layout

# Khái niệm

Trong ngôn ngữ lập trình C, bộ nhớ của chương trình thường được phân chia thành các phân vùng khác nhau, mỗi phân vùng có mục đích và quy tắc sử dụng riêng. Dưới đây là một số phân vùng nhớ quan trọng trong C:

1. **Text Segment**

Text Segment là một phần của bộ nhớ trong mô hình bộ nhớ của một chương trình máy tính, được sử dụng để lưu trữ mã máy của chương trình. Cụ thể, Text Segment chứa mã máy đã được biên dịch từ mã nguồn của chương trình và được CPU thực thi để thực hiện các hành động quy định trong chương trình.

Một số đặc điểm:

* Mã máy:
* Chứa mã máy, tức là tập hợp các lệnh thực thi được CPU hiểu được.
* Mã máy được tạo ra thông qua quá trình biên dịch từ mã nguồn của chương trình.
* Quyền truy cập:
* Text Segment thường có quyền đọc và thực thi, nhưng không có quyền ghi. điều này ngăn chặn chương trình việc tự sửa đổi mã máy của nó.
* Kích thước cố định:
* Kích thước của Text Segment thường là cố định khi chương trình được biên dịch.
* Điều này giúp hệ điều hành và CPU dễ dàng quản lý vùng nhớ này.
* Lưu Trữ Code và Hằng Số:
* Bao gồm cả mã máy của các hàm và các hằng số.
* Các hàm được gọi từ các phần khác của chương trình sẽ có địa chỉ trong Text Segment.

1. **Data Segment**

Data Segment là một phần của bộ nhớ trong mô hình bộ nhớ của một chương trình máy tính, được sử dụng để lưu trữ dữ liệu tĩnh của chương trình. Dữ liệu tĩnh bao gồm biến toàn cục và biến tĩnh (static), tức là các biến mà không phụ thuộc vào thời gian chạy của chương trình.

Một số đặc điểm:

* Biến Toàn Cục (Global Variables):
* Chứa giá trị của các biến toàn cục được khai báo trong chương trình.
* Các biến này có thể được truy cập từ bất kỳ hàm nào trong chương trình.
* Biến Tĩnh (Static Variables):
* Chứa giá trị của các biến tĩnh, nghĩa là biến được khai báo với từ khóa static.
* Có thể được truy cập chỉ trong phạm vi của hàm mà chúng được khai báo.
* Initialized Data Segment (Dữ liệu Đã Khởi Tạo):
* Chứa giá trị của các biến toàn cục và biến tĩnh được khởi tạo với giá trị ban đầu.
* Dữ liệu này được sao chép từ bộ nhớ của chương trình thực thi.
* Uninitialized Data Segment (Dữ liệu Chưa Khởi Tạo):
* Chứa giá trị mặc định của các biến toàn cục và biến tĩnh mà không cần khởi tạo giá trị ban đầu.
* Dữ liệu trong phân vùng này thường được xác định bởi giá trị 0.
* Quyền truy cập: Data Segment thường có quyền đọc và ghi, nghĩa là dữ liệu có thể được đọc và sửa đổi trong quá trình thực thi của chương trình.
* Kích thước cố định: Kích thước của Data Segment có thể thay đổi trong quá trình thực thi của chương trình khi các biến được khởi tạo hoặc giải phóng.

1. **Heap**

Heap là một phần của bộ nhớ trong mô hình bộ nhớ của một chương trình máy tính, được sử dụng để cấp phát bộ nhớ động. Các biến được cấp phát trên heap không có kích thước xác định tại thời điểm biên dịch và có thể được quản lý động trong quá trình thực thi của chương trình.

Một số đặc điểm:

* Cấp phát động:
* Heap được sử dụng để cấp phát bộ nhớ động trong quá trình thực thi của chương trình.
* Điều này cho phép chương trình tạo ra và giải phóng bộ nhớ theo nhu cầu, thích ứng với sự biến đổi của dữ liệu trong quá trình chạy.
* Quyền truy cập: Bộ nhớ trên heap thường có quyền đọc và ghi, nghĩa là dữ liệu có thể được đọc và sửa đổi trong suốt thời gian chương trình chạy.
* Cấp Phát và Giải Phóng Bộ Nhớ: Các hàm như malloc(), calloc(), realloc(), và free() được sử dụng để cấp phát và giải phóng bộ nhớ trên heap.
* Kích Thước Thay Đổi: Kích thước của heap có thể thay đổi trong quá trình thực thi của chương trình, tùy thuộc vào các thao tác cấp phát và giải phóng bộ nhớ.
* Không Giữ Giá Trị Mặc Định: Bộ nhớ trên heap không giữ giá trị mặc định như trong Data Segment. Nếu không được khởi tạo, giá trị của biến trên heap sẽ không xác định.

Ví dụ cấp phát động:

| #**include** <stdio.h> #**include** <stdlib.h>  **int** **main**() {  // Cấp phát bộ nhớ động cho một biến kiểu int  **int** \*dynamicVar = (**int** \*)malloc(**sizeof**(**int**));   **if** (dynamicVar != NULL) {  // Gán giá trị cho biến trên heap  \*dynamicVar = 42;   // In giá trị  printf("Value on heap: %d\n", \*dynamicVar);   // Giải phóng bộ nhớ trên heap  free(dynamicVar);  }   **return** 0; } |
| --- |

1. Stack

Stack là một phần quan trọng của bộ nhớ trong mô hình bộ nhớ của một chương trình máy tính. Nó được sử dụng để lưu trữ các biến cục bộ, các giá trị trả về từ hàm, địa chỉ trả về và một số thông tin khác liên quan đến thực thi của chương trình.

Một số đặc điểm:

* LIFO: Stack hoạt động theo nguyên tắc LIFO, nghĩa là phần tử cuối cùng được thêm vào sẽ là phần tử đầu tiên được lấy ra.
* Các biến cục bộ: Chứa các biến cục bộ, tức là các biến được khai báo trong các hàm và chỉ có giá trị trong phạm vi của hàm đó.
* Kích thước cố định: Tùy thuộc vào hệ điều hành khác nhau
* Địa chỉ trả về: Khi một hàm được gọi, địa chỉ trả về (return address) được đẩy vào ngăn xếp để theo dõi nơi chương trình sẽ tiếp tục sau khi hàm kết thúc thực thi.
* Thực hiện gọi hàm và trả về: Khi một hàm được gọi, các giá trị và địa chỉ quan trọng liên quan đến hàm đó được đẩy vào ngăn xếp. Sau khi hàm kết thúc, những giá trị này được lấy ra để quay trở lại thực thi chương trình.
* Khối lệnh ( Frame): Mỗi hàm khi được gọi tạo ra một khối lệnh (frame) trên ngăn xếp, chứa các biến cục bộ và các giá trị khác liên quan đến hàm.
* Stack pointer: Con trỏ ngăn xếp (stack pointer) giữ địa chỉ hiện tại của stack và được cập nhật liên tục khi dữ liệu được đẩy vào hoặc lấy ra.

Stack là một cơ chế quan trọng trong quá trình thực thi chương trình và thường được sử dụng để quản lý luồng thực thi, gọi hàm, và theo dõi các biến cục bộ. Các ngôn ngữ lập trình thường sử dụng ngăn xếp để triển khai thực thi hàm và quản lý các biến cục bộ.

1. Memory - mapped Segment

Memory-mapped segment là một phần của bộ nhớ trong mô hình bộ nhớ của một chương trình máy tính, được sử dụng để tương tác với các thiết bị ngoại vi thông qua các địa chỉ bộ nhớ. Thông thường, các thiết bị ngoại vi như cổng vào/ra (I/O ports) và thanh ghi của các thiết bị được ánh xạ trực tiếp vào không gian địa chỉ bộ nhớ của chương trình.

Một số đặc điểm:

* Tương tác với thiết bị ngoại vi: Memory-mapped segment cho phép chương trình tương tác với các thiết bị ngoại vi bằng cách truy cập địa chỉ bộ nhớ tương ứng với các thanh ghi và cổng I/O của các thiết bị đó.
* Địa chỉ bộ nhớ ánh xạ: Các địa chỉ bộ nhớ của các thanh ghi và cổng I/O của các thiết bị ngoại vi được ánh xạ vào không gian địa chỉ bộ nhớ của chương trình.
* Quyền truy cập: Memory-mapped segment thường có thể có quyền đọc và ghi để cho phép chương trình gửi dữ liệu đến thiết bị và đọc dữ liệu từ thiết bị.
* Giao tiếp với thiết bị ngoại vi: Thông qua Memory-mapped segment, chương trình có thể gửi lệnh và nhận dữ liệu từ các thiết bị ngoại vi, như cổng COM, bàn phím, màn hình và các thiết bị khác.
* Sử dụng trong ngôn ngữ mã máy và assembly: Memory-mapped segment thường được sử dụng nhiều trong ngôn ngữ máy và ngôn ngữ gần với phần cứng để thực hiện các thao tác gần với phần cứng của hệ thống.
* Điều khiển thiết bị: Ánh xạ địa chỉ bộ nhớ của các thanh ghi và cổng I/O của thiết bị ngoại vi cho phép chương trình kiểm soát và điều khiển các thiết bị ngoại vi.

Lưu ý rằng sử dụng Memory-mapped segment đôi khi yêu cầu quyền hệ điều hành và có thể chỉ có sẵn trong môi trường thực thi nào đó (ví dụ như hệ điều hành Unix-like).

# Stack và Heap

Bộ nhớ Heap và bộ nhớ Stack bản chất đều cùng là vùng nhớ được tạo ra và lưu trữ trong RAM khi chương trình được thực thi.

Bộ nhớ Stack được dùng để lưu trữ các biến cục bộ trong hàm, tham số truyền vào... Truy cập vào bộ nhớ này rất nhanh và được thực thi khi chương trình được biên dịch.

Bộ nhớ Heap được dùng để lưu trữ vùng nhớ cho những biến con trỏ được cấp phát động bởi các hàm *malloc - calloc - realloc* (trong C).

Kích thước vùng nhớ:

* Stack: kích thước của bộ nhớ Stack là cố định, tùy thuộc vào từng hệ điều hành, ví dụ hệ điều hành Windows là 1 MB, hệ điều hành Linux là 8 MB (lưu ý là con số có thể khác tùy thuộc vào kiến trúc hệ điều hành của bạn).
* Heap: kích thước của bộ nhớ Heap là không cố định, có thể tăng giảm do đó đáp ứng được nhu cầu lưu trữ dữ liệu của chương trình.

Đặc điểm vùng nhớ:

* Stack: vùng nhớ Stack được quản lý bởi hệ điều hành, dữ liệu được lưu trong Stack sẽ tự động hủy khi hàm thực hiện xong công việc của mình.
* Heap: Vùng nhớ Heap được quản lý bởi lập trình viên (trong C hoặc C++), dữ liệu trong Heap sẽ không bị hủy khi hàm thực hiện xong, điều đó có nghĩa bạn phải tự tay hủy vùng nhớ bằng câu lệnh free (trong C), và delete hoặc delete [] (trong C++), nếu không sẽ xảy ra hiện tượng rò rỉ bộ nhớ.

Vấn đề lỗi xảy ra với vùng nhớ:

* Stack: bởi vì bộ nhớ Stack cố định nên nếu chương trình bạn sử dụng quá nhiều bộ nhớ vượt quá khả năng lưu trữ của Stack chắc chắn sẽ xảy ra tình trạng tràn bộ nhớ Stack (Stack overflow), các trường hợp xảy ra như bạn khởi tạo quá nhiều biến cục bộ, hàm đệ quy vô hạn,...

Ví dụ tràn bộ nhớ stack khi gọi hàm đệ quy vô hạn:

| **int** **foo**(**int** x){  printf("De quy khong gioi han\n");  **return** foo(x); } |
| --- |

* Heap: Nếu bạn liên tục cấp phát vùng nhớ mà không giải phóng thì sẽ bị lỗi tràn vùng nhớ Heap (Heap overflow). Nếu bạn khởi tạo một vùng nhớ quá lớn mà vùng nhớ Heap không thể lưu trữ một lần được sẽ bị lỗi khởi tạo vùng nhớ Heap thất bại.

Ví dụ trường hợp khởi tạo vùng nhớ Heap quá lớn:

| **int** \*A = (**int** \*)malloc(18446744073709551615); |
| --- |