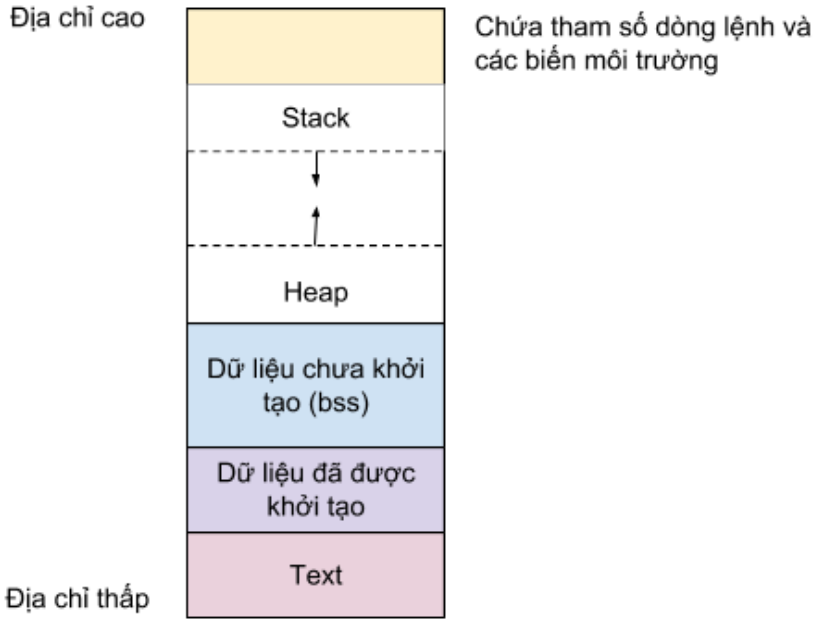
Sơ đồ bộ nhớ trong các chương trình C

Hiểu được sơ đồ, kiến trúc các vùng nhớ của một chương trình sẽ giúp bạn viết code một các hiệu quả và tối ưu. Mỗi vùng nhớ trong một chương trình lại có tính chất khác nhau. Về cơ bản trong một chương trình C, bộ nhớ được biểu hiện bởi các vùng sau:

- Vùng Text
- Vùng dữ liệu đã được khởi tạo
- Vùng dữ liệu chưa được khởi tạo
- Vùng Stack
- Vùng Heap



Chương trình cơ sở

Xét đoạn code sau:

```
#include <stdio.h>

int main()
{
   return 0;
}
```

Chương trình không chứa bất kỳ một biến nào, nên ta sẽ sử dụng nó để so sánh sự thay đổi kích thước trong vùng nhớ bằng lệnh size(1). Lệnh size(1) cho ta biết kích thước (byte) của các vùng text, DS, BSS.

Kiểm tra kích thước các vùng nhớ bằng cách lưu đoạn code trên vào 1 file source code "test.c" và làm như sau:

```
$ gcc test.c -o memory-layout-no-var

$ size memory-layout-no-var

  text        data        bss    dec      hex    filename

  1115        552         8      1675     68b    memory-layout-no-var
```

Chú ý: Trong phạm vi bài viết này chúng tôi chỉ hướng dẫn cách kiểm tra kích thước các vùng nhớ trên hệ điều

hành Linux (ubuntu, centos ...). Nếu các bạn đang dùng windown có thể dùng linux online terminal[1]

Vùng Text

Các tên khác: text segment hay code segment.

Vùng này chứa mã máy của chương trình đã được biên dịch.

Vùng text segment của một đối tượng file có thể thực thi được thường là vùng nhớ chỉ đọc, điều này là để tránh việc chương trình vô tình bị sửa đổi không như mong muốn.

Thông thường vùng này là vùng nhớ chia sẻ nên ta chỉ cần một phiên bản sao chép tồn tại trong bộ nhớ cho những chương trình thực thi thường xuyên.

Vùng dữ liệu đã được khởi tạo

Các tên khác: Initialized data segment, data segment (DS)

Để cho ngắn gon ta sẽ sử dụng "DS" như tên của vùng này.

DS là một phần trong bộ nhớ ảo của chương trình, nơi chứa những biến toàn cục, biến tĩnh hoặc các biến hằng đã được khởi tạo trong code.

Trong ví dụ sau các biến a và b sẽ được chứa trong vùng nhớ này.

```
#include <stdio.h>

int a = 9; /* Khởi tạo giá trị cho biến toàn cục a */
static int b = 2; /* Khởi tạo giá trị cho biến toàn cục b */
const int c = 8;  /* Biến hằng c được chứa trong vùng nhớ khởi tạo chỉ đọc */

int main()
{
    b = a + 1;
    printf("%d %d %d\n", a, b, c);
    return 0;
}
```

Chú ý răng DS không phải là vùng nhớ chỉ đọc, bởi vì giá trị của các biến có thể được thay đổi khi chạy runtime.

Thực ra vùng nhớ này có thể chia ra làm hai vùng nhỏ hơn, vùng khởi tạo chỉ đọc dùng để chứa các biến hăng, và vùng khởi tạo đọc-ghi để chứa các biến toàn cục và biến tĩnh thông thường.

Ta lưu đoạn code trên vào file "ds.c", thực hiện biên dịch bằng gcc và kiểm tra kích thước của chương trình trong các vùng nhớ bằng lệnh size để thấy được sự thay đổi giá trị của các vùng nhớ so với chương trình cơ sở ở trên.

```
$ gcc ds.c -o memory-layout-ds

$ size memory-layout-ds

  text          data          bss    dec          hex        filename

  1250          568           8      1826         722        memory-layout-ds
```

Nhận thấy chỉ có kích thước của vùng text (do lượng code logic tăng lên) và data thay đổi so với chương trình cơ sở. điều này chứng tỏ các biến a, b, c ở trên đã được lưu vào vùng dữ liệu đã được khởi tạo.

Vùng dữ liệu chưa được khởi tạo

Các tên khác: Uninitialized Data Segment hay vùng BSS

Dữ liệu trong vùng này được khởi tạo bởi nhân của hệ điều hành về giá trị 0 trước khi chương trình được thực sự thực thi.

Vùng này chứa các biến toàn cục và các biến tĩnh được khởi tạo với các giá trị 0 (zero) hoặc không được khởi tạo với bất kỳ giá trị nào trong source code.

Trong ví dụ sau các biến global và i sẽ được chứa trong vùng nhớ bss.

```
#include <stdio.h>

int global; /* Biến toàn cục chưa khởi tạo được chứa trong bss*/

int main(void)
{
    static int i; /* Biến tĩnh chưa khởi tạo được chứa trong bss */
    return 0;
}
```

Lưu chương trình vào file bss.c và thực hiện như sau để kiểm tra sự thay đổi kích thước cách vùng nhớ so với chương trình cơ sở.

```
$ gcc bss.c -o memory-layout-bss

$ size memory-layout-bss

  text          data          bss   dec      hex     filename

  1115          552           16    1683     693     memory-layout-bss
```

Trong trường hợp này chỉ có vùng bss thay đổi kích thước so với chương trình cơ sở, vậy nên kết luận các biến chưa được khởi tạo "global" và "i" được lưu trong vùng bss là chính xác.

Vùng nhớ Stack

Vùng nhớ stack thường nằm liền kề với vùng heap. Hai vùng nhớ này phát triển theo chiều đối nhau. Ví trí đỉnh của stack được đánh dấu bằng stack pointer, vị trí đỉnh của heap được đánh đấu bằng heap pointer. Khi hai con trỏ này gặp nhau cũng là lúc bộ nhớ dành cho chúng bị cạn kiệt.

Tuy nhiên bạn không cần quá lo lắng về vấn đề ở trên vì ngày nay phần cứng của bộ nhớ (RAM vật lý) đã được tăng lên, hơn thế nữa người ta cũng áp dụng những kỹ thuật đánh địa chỉ ảo (bộ nhớ ảo), làm cho hai vùng này lớn hơn rất nhiều.

Vùng nhớ stack tuân theo cấu trúc LIFO (Last In First Out)và có một thanh ghi stack pointer để lưu giữ vị trí đỉnh của nó.

Các loại dữ liệu được chứa trong vùng nhớ này bao gồm:

- Biến cục bộ, biến tạm.
- Địa chỉ trả về sau mỗi lời gọi hàm.
- Các tham số truyền vào các hàm.

Các thuật toán đệ quy được thực hiện dựa trên hoạt động của stack và chúng được áp dụng rất nhiều trong thực tế.

Lỗi có thể gặp với vùng nhớ này chính là "stack overflow" khi mà ta sử dụng đệ quy không hồi kết hoặc đệ quy quá sâu. Nó cũng có thể xảy ra khi ta cấp phát một lượng bộ nhớ quá lớn, vượt quá giới hạn của stack.

Các biến tạo trong vùng nhớ này có thể tự động được thu hồi, nên ta không cần các kỹ thuật thu hồi bộ nhớ như khi ta cấp phát động trong heap.

Vùng nhớ Heap

Ngoài các đặc tính đã được đề cập ở trên vùng nhớ heap còn có một số đặc tính sau:

Các biến chứa trong vùng nhớ này phải được thu hồi thủ công bằng cách sử dụng các hàm delete, delete [] và free trong C/C++.

Việc cấp phát bộ nhớ cho một biến trong vùng này là châm hơn so với việc cấp phát trong stack. Có thể giải thích lý do này là vì vùng nhớ trong stack là luôn luôn liên tục nên việc cấp phát dễ dàng hơn nhiều.

Lỗi phân mảnh bộ nhớ hay xảy ra với bộ nhớ này qua thời gian sử dụng lâu, vì ta đã thực hiện nhiều thao tác cấp phát và thu hồi bộ nhớ trong nó, để lại nhiều vùng nhớ trống vô cùng nhỏ, khó xử dụng được.

Trong C/C++ có thể cấp phát trong vùng nhớ này qua các hàm new, malloc, calloc … Việc cấp phát có thể không thành công do lượng cấp phát vượt quá giới hạn của heap.

Vùng nhớ này thường được chỉa sẻ bởi các thư viện chia sẻ (shared libraries) hoặc các mô đun được tải động trong các tiến trình.

Hiểu về cách thức hoạt động, tính chất của các vùng nhớ là vô cùng quan trong với các lập trình viên viết các ngôn ngữ bậc thấp như C/C++, Assembly.

Tuy nhiên nó cũng có ích với bất cứ lập trình viên nào để giúp họ cải thiện được tốc độ, năng lực của chương trình, giúp tiết kiệm tài nguyên phần cứng và chương trình chạy hiệu quả hơn. Bạn có thể tìm hiểu thêm về các cấu trúc dữ liệu stack và heap để hiểu sâu hơn về cách hoạt động của hai vùng nhớ quan trọng này.