Cấp phát bộ nhớ động trong C : Malloc hay Calloc

share
writes
tutorial
c

Dung_Nguyen (Coulson) 2015-04-24 09:01:33 UTC #1

Cấp phát bộ nhớ động trong C : Malloc vs Calloc

Để cấp phát bộ nhớ động trong C, chúng ta có 2 cách:

1. `void* malloc (size_t size);`
2. `void* calloc (size_t num, size_t size);`

So sánh

Tiêu chuẩn	malloc	calloc
Cú pháp	`void* malloc (size_t size);`	`void* calloc (size_t num, size_t size);`
Mục đích	cấp phát một vùng nhớ có kích thước	cấp phát vùng nhớ đủ chứa **num** phần tử,
	là **size**	mỗi phần tử có kích thước là **size**
Số tham số	1	2
Kết quả trả về	Con trỏ tới vùng nhớ được cấp phát nếu	Con trỏ tới vùng nhớ được cấp phát nếu
	thành công, con trỏ NULL nếu không thành,công	
Giá trị khởi tạo	Giá trị rác	Được gán bằng 0

Sử dụng

- Khi sử dụng `malloc` phải tính toán kích thước vùng nhớ cần cấp phát trước rồi truyền vào cho `malloc`
- Khi sử dụng `calloc` chỉ cần truyền vào số phần tử và kích thước 1 phần tử, thì `calloc` sẽ tự động tính toán và cấp phát vùng nhớ cần thiết

Ví dụ: Cấp phát mảng 10 phần tử kiểu int:

```
int *a = (int *) malloc( 10 * sizeof( int ));
int *b = (int *) calloc( 10, sizeof( int ));
```

Hiệu suất / Perfomance

malloc nhanh hơn so với **calloc**. Lý do là **calloc** ngoài việc có nhiệm vụ cấp phát vùng nhớ như **malloc**, nó còn phải gán giá trị cho tất cả các phần tử của vùng nhớ vừa cấp phát = 0

```
int *a = (int *) calloc(10, sizeof( int ));
```
tương đương với
```
int *b = (int *) malloc( 10 * sizeof( int ));
memset(b, 0, 10 * sizeof(int));
```

Sự an toàn

Sử dụng **calloc** an toàn hơn **malloc** vì sau khi khởi tạo vùng nhớ thì **calloc** sẽ khởi tạo vùng nhớ cấp phát = 0, còn vùng nhớ do **malloc** cấp phát vẫn chứa giá trị rác nên sẽ dễ gây ra lỗi nếu truy xuất tới vùn nhớ này trước khi gán cho nó một giá trị xác định.

Quản lý bộ nhớ và vấn đề về new,delete trong C++

Thắc mắc về cấp phát động và malloc, calloc trong C!

Code bị lỗi khiến file *.exe bị dừng

---

ltd (Lê Trần Đạt) 2015-04-24 09:39:41 UTC #2

> Dung_Nguyen:
>
> Sự an toàn
>
>] Sử dụng calloc an toàn hơn malloc vì sau khi khởi tạo vùng nhớ thì calloc sẽ khởi tạo vùng nhớ cấp phát = 0, còn vùng nhớ do malloc cấp phát vẫn chứa giá trị rác nên sẽ dễ gây ra lỗi nếu truy xuất tới vùn nhớ này trước khi gán cho nó một giá trị xác định.

Bài viết hay, nhưng không đồng ý với ý này. Nên sửa lại là chọn `malloc` hay `calloc` là do mục tiêu sử dụng. Nếu mình không quan tâm đến giá mặc định của vùng nhớ được cấp thì dùng `malloc` còn nếu muốn tất cả là `0` thì dùng `calloc`.

Như vậy hợp lý hơn, đưa lựa chọn vào tay người code. Còn nếu viết về sự an toàn thì sẽ đẩy người đọc về phía chọn `calloc` mặc dù không phải lúc nào cũng cần thiết.

Góp ý một tí về trình bày, tại sao **@Dung_Nguyen** lại để thêm một dấu] ở đầu một số dòng vậy?

Mark (Trịnh Minh Cường) 2015-04-24 10:50:10 UTC #3

em toàn dùng new và delete 😄 , dùng lai luôn của C++ 😄

Dung_Nguyen (Coulson) 2015-04-24 10:54:10 UTC #4

Thanks bác Đạt. em mới sửa lại title.
Còn dấu [là do lúc em replace all trong markdown nó replace lộn chút

---

Dung_Nguyen (Coulson) 2015-04-24 11:12:24 UTC #5

hôm trước phỏng vấn VNG nó còn hỏi sự khác nhau giữa malloc với new nữa mà không trả lời được nên mới phải về tìm hiểu lại. 😮

Mark (Trịnh Minh Cường) 2015-04-24 11:14:50 UTC #6

😄 Cái này em cũng chưa biết tại em thấy new thì cú pháp nó gọn, ngắn hơn 2 thằng kia nên dùng lai luôn 😄

Ten_Gi (Tên Gì) 2015-07-20 02:45:27 UTC #7

New chỉ có trong C++ còn cái này là C mà new phải đi vs realloc chứ

Dung_Nguyen (Coulson) 2015-07-20 07:14:59 UTC #8

realloc dùng để thay đổi/ xóa vùng nhớ đã được cấp phát còn malloc và calloc dùng để cấp phát vùng nhớ mới.

---

programmerit (Chi Ngo) 2015-07-20 09:45:04 UTC #9

> Dung_Nguyen:
>
> malloc nhanh hơn so với calloc. Lý do là calloc ngoài việc có nhiệm vụ cấp phát vùng nhớ như malloc, nó còn phải gán giá trị cho tất cả các phần tử của vùng nhớ vừa cấp phát = 0

Tại sao không phải vì calloc phải làm 10 lần công việc của malloc nhỉ? Bạn giải thích cho mình với?

Dung_Nguyen (Coulson) 2015-07-21 02:34:02 UTC #10

> Dung_Nguyen:
>
> ```
> int *a = (int *) calloc(10, sizeof( int ));
> ```
> tương đương với
>
> ```
> int *b = (int *) malloc( 10 * sizeof( int ));
> memset(b, 0, 10 * sizeof(int));
> ```

Như bạn thấy thì calloc làm 2 bước:
1. Cấp phát bộ nhớ giống như malloc
2. Khởi tạo giá trị vùng nhớ = 0 tương tự như lúc sử dụng hàm memset

Nên là nó không thể gấp 10 lần được.
Vậy tại sao bạn lại nghĩ là calloc làm 10 lần công việc của malloc?

---

programmerit (Chi Ngo) 2015-07-21 02:58:13 UTC #11

> Dung_Nguyen:
>
> Như bạn thấy thì calloc làm 2 bước:1. Cấp phát bộ nhớ giống như malloc2. Khởi tạo giá trị vùng nhớ = 0 tương tự như lúc sử dụng hàm memset

Thế mình lấy một ví dụ đơn giản như thế này nhé:
Mình muốn tính tổng từ 1 đến 10.
Mình sẽ có hai hàm như sau:

```
int tonghaso(int a, int b) {
     return a + b;
}
```

```
int tong(int start, int end) {
    int s  = 0;
    for( int i = start;i <=end;i++) {
          s+=i;
    }
    return s;
}
```

Như vậy, nếu mình tính tổng từ 1 đến 10 thì mình cần dùng 9 lần hàm `tonghaiso` và 1 lần hàm `tong`. Nhưng ở đây nếu không tính đến việc gọi hàm mà chỉ tính đến số phép tính để thực hiện được yêu cầu thì rõ ràng là như nhau đúng không (9 phép tính)?
Không phải vì gọi hàm một lần mà nhanh hơn so với gọi 2 hàm khác như bạn đưa ra. Nhỡ đâu ở trong cái phần một hàm ấy, người ta làm rất nhiều việc thì sao? Mà về bản chất ở đây là việc cấp phát bộ nhớ, có thể hình dung bạn có 1 cái bánh mỳ (ở BigC) chẳng hạm, một cái bạn chia thành 10 phần thì bạn phải cắt 9 lần, trong khi cái còn lại thì bạn không phải cắt mà chỉ đánh dấu lên nó 10 vách thôi (hi vọng việc đánh dấu sẽ nhanh hơn việc cắt).

Rok_Hoang (Minh Hoàng) 2015-07-21 03:06:26 UTC #12

> calloc is **a tiny bit slower** than malloc because of the extra step of initializing the memory region allocated. However, in practice the difference in speed is very small and can be ignored.
> **http://www.diffen.com/difference/Calloc_vs_Malloc**

Theo mình thì nên sử dụng calloc vì không nên để giá trị trong biến mà mình không kiểm soát. Tuy nhiên cũng tùy mục đích sử dụng.

Dung_Nguyen (Coulson) 2015-07-21 03:26:20 UTC #13

> programmerit:
>
> Không phải vì gọi hàm một lần mà nhanh hơn so với gọi 2 hàm khác như bạn đưa ra. Nhỡ đâu ở trong cái phần một hàm ấy, người ta làm rất nhiều việc thì sao? Mà về bản chất ở đây là việc cấp phát bộ nhớ, có thể hình dung bạn có 1 cái bánh mỳ (ở BigC) chẳng hạm, một cái bạn chia thành 10 phần thì bạn phải cắt 9 lần, trong khi cái còn lại thì bạn không phải cắt mà chỉ đánh dấu lên nó 10 vách thôi (hi vọng việc đánh dấu sẽ nhanh hơn việc cắt).

1. Như mình đã nói, `calloc` sẽ chậm hơn `malloc` vì nó phải set bộ nhớ = 0.
2. Việc chia bánh mỳ như bạn nói mình không thấy nó tương tự như việc cấp phát bộ nhớ.
   Nếu như bạn muốn lấy 1/2 cái bánh mỳ:
3. calloc tương tự như việc cắt 1/2 cái bánh mì và trét bơ lên đó.
4. malloc tương tự như việc chỉ mới chia đôi cái bánh mì và lấy 1/2. bạn phải làm thêm bước trét bơ nữa.

H_ng_Arsenal (Hưng Arsenal) 2016-02-09 17:56:10 UTC #14

Bạn có thể đề cập đến hàm realloc được không?

Doi_Ten (Đổi Tên) 2016-04-18 13:40:22 UTC #15

anh có thể cho em hỏi là tại sao phải cấp phát cho con trỏ k? nếu k cấp phát thì ảnh hưởng j ?

ThuyChamChap (Thu Thuỷ) 2016-05-05 05:18:00 UTC #16

con trỏ là biến đặc biệt, chỉ trỏ tùm lum, có thể lưu trữ nhiều giá trị (mảng), nên mỗi giá trị phải có 1 cái lỗ chứa ==> Phải cấp phát bộ nhớ

---

tcm (Người bí ẩn) 2016-05-26 07:56:55 UTC #17

> Dung_Nguyen:
>
> realloc dùng để thay đổi/ xóa vùng nhớ đã được cấp phát

Theo mình biết thì đó chỉ là 1 trong 2 công dụng của `realloc` thôi mà?
Ngoài tác dụng thay đổi/xóa vùng nhớ đã được cấp phát thì `realloc` còn có thể Tạo mới vùng nhớ đối với những vùng nhớ chưa được khởi tạo mà ?
VD: (cấp phát bộ nhớ cho 2 con trỏ a và b)

```
int *a, *b;
a = (int *)realloc(0, sizeof(int *));
b = (int *)realloc(0, sizeof(int *));
```

P/S: Sao bạn không nói đến giải phóng con trỏ `free` (hay `delete`) luôn ? (vì mình thấy nó có liên quan đến chủ đề Topic này) 😃

Peehus (Nguyen Sy Phu) 2016-12-17 18:26:21 UTC #18

```
int *a = (int *) malloc( 10 * sizeof( int ));
int *b = (int *) calloc( 10, sizeof( int ));
```

Cho em hỏi (int *) là ép kiểu như thế nào vậy ạ?

---

tcm (Người bí ẩn) 2016-12-17 23:22:00 UTC #19

> Peehus:
>
> Cho em hỏi (int *) là ép kiểu như thế nào vậy ạ?

2 hàm malloc và calloc đều có kiểu `void *`, mà `void *` thì bạn phải ép kiểu cho nó đúng với kiểu dữ liệu của con trỏ 🙂

---

trongle98 (Lê Văn Trọng) 2017-05-21 06:18:51 UTC #20

Cho em hỏi nếu mình khai báo như thế này : int* x = (int *)malloc(sizeof(int)); với một biến kiểu int có kích thước 4 bytes thì nó chỉ sử dụng có 4 bytes bộ nhớ đúng không ạ?

---

hoangthan (Thân Hoàng) 2017-09-15 09:02:31 UTC #21

Mọi người cho em hỏi về hàm malloc cụ thể của code này:

```
1. main()
2. {
3. char *ten;
4. ten = malloc(1);
5. gets(ten);
6. }
```

malloc(1) được cấp phát 1 byte bộ nhớ, kiểu char có kích thước 1 byte. Nhưng em có thể nhập được rất nhiều kí tự: khoảng hơn 200 kí tự. Ai giải thích giúp em với 😕

---

rogp10 (rogp10) 2017-09-15 09:34:09 UTC #22

> hoangthan:
>
> malloc(1) được cấp phát 1 byte bộ nhớ, kiểu char có kích thước 1 byte. Nhưng em có thể nhập được rất nhiều kí tự: khoảng hơn 200 kí tự. Ai giải thích giúp em với 😕

Nhiều khi thực nhận từ heap là mấy trăm byte ấy chứ không phải 1 byte. Còn mảng tĩnh mà nhập vậy là hỏng hết.

---

hoangthan (Thân Hoàng) 2017-09-15 09:43:31 UTC #23

em vẫn chưa thông cho lắm. Em muốn hỏi tại sao nó chỉ được cấp 1 byte để dùng mà nó lại chứa được rất nhiều ký tự char ấy 😕

---

try_forever (nghia) 2017-09-15 09:45:30 UTC #24

mấy hôm trước em học lập trình thread basic trên youtube: thì học đc khởi tạo vùng nhớ khá bất ngờ em còn tưởng là sai nhưng ôi trời ơi đó là đúng `struct SinhVien* sv = malloc(*sv);` em còn dựt mình! Người đó giải thích = tiếng anh em chưa hiểu lắm! Anh nào có thể giải thích giúp em không?

---

rogp10 (rogp10) 2017-09-15 09:47:56 UTC #25

> hoangthan:
>
> em vẫn chưa thông cho lắm. Em muốn hỏi tại sao nó chỉ được cấp 1 byte để dùng mà nó lại chứa được rất nhiều ký tự char ấy 😕

Cái này là nội vụ (internal) của C 🙂 bạn tìm malloc implementation nhé.

> try_forever:

> mấy hôm trước em học lập trình thread basic trên youtube: thì học đc khởi tạo vùng nhớ khá bất ngờ em còn tưởng là sai nhưng ôi trời ơi đó là đúng `struct SinhVien* sv = malloc(*sv);` em còn dựt mình! Người đó giải thích = tiếng anh em chưa hiểu lắm! Anh nào có thể giải thích giúp em không?

Bạn xem lại xem có thiếu từ gì không đã.

---

try_forever (nghia) 2017-09-15 09:50:07 UTC #26

ok anh em thiếu `sizeof(*sv)` đây là link: **https://youtu.be/KVF4kYvd7e8?t=2m37s**

---

hoangthan (Thân Hoàng) 2017-09-15 09:53:01 UTC #27

> rogp10:
>
> malloc implementation

struct Sinhvien *sv;
thì sv thực chất chỉ là một biến(con trỏ chứa biến dạng cấu trúc).
Hàm malloc cung cấp cho con trỏ sv một lượng bộ nhớ bằng độ lớn của cả cấu trúc sv đó.

---

hoangthan (Thân Hoàng) 2017-09-15 09:58:36 UTC #28

The *structure-name* part of the definition may be omitted:

```
struct {
    char    name[30];   /* name of the part */
    int     quantity;   /* how many are in the bin */
    int     cost;       /* The cost of a single part  (in cents) */
} printer_cable_bin;    /* where we put the print cables */
```

The variable `printer_cable_bin` has been defined, but no data type has been created. In other words, `printer_cable_bin` is the only variable of this structure you want in the program. The data type for this variable is an*anonymous structure*.

---

try_forever (nghia) 2017-09-15 10:50:38 UTC #31

Em tán thành với ý kiến của anh! sài printf, scanf thật là hack não người học ....

Tran_Quang_Son (Trần Quang Sơn) 2017-11-11 00:00:20 UTC #33

em muốn có một chương trình kiểu kệ cha người dùng nhập bao nhiêu phần tử trong mảng mà không muốn ước lượng số phần tử, thì dùng mảng động có đc không và dùng ntn, như trên thì hình như phải biết số phần tử

---

rogp10 (rogp10) 2017-11-11 01:18:22 UTC #34

Không hẳn, tùy vào HĐH nữa. Linux nó lazy cho 1 page toàn zero, tức là khi viết vào đó sẽ bị văng lỗi và lúc này mới được cấp mem thật sự (gọi là overcommit).

https://www.etalabs.net/overcommit.html

---

noname00 (HK boy) 2017-11-11 11:24:46 UTC #37

> sycoi001:
>
> auto x=0.0

> sycoi001:
>
> C++ có cái vector sẽ hỗ trợ cái bạn nói.

Bạn cmt cấu trúc C++ vào topic về C làm gì?

---

rogp10 (rogp10) 2017-12-28 10:21:03 UTC #38

`calloc` có thể xem là an toàn hơn ở một điểm: kích cỡ của ô nhớ và số ô nhớ là hai tham số khác nhau 🙂

---

quangquangvu (Quang Vu Quang) 2017-12-29 06:28:32 UTC #39

Ngày xưa khi còn chạy DOS, bộ nhớ hạn hẹp nên người ta phải tối ưu bộ nhớ bằng malloc/calloc.
Giờ máy bèo bèo cũng 4GB, trung bình cũng 8GB chưa kể vùng swap. Nhiều ngôn ngữ lập trình như php/javascript còn . . . chả quan tâm đến vấn đề ít hay nhiều memory.

Nhưng lâu lâu nghe lại cũng vui (y) !

Dark.Hades (Dark.Hades) 2017-12-29 06:33:08 UTC #40

> quangquangvu:
>
> php

PHP vẫn care bác nhé, điển hình như một số thay đổi của yield, foreach,...
Có thằng Python/Ruby/JS là mất dạy nhất thôi. Thằng Java có khi cũng không kém cạnh, cứ kêu byte code nhưng ăn hết cả ram nhà người ta :))
