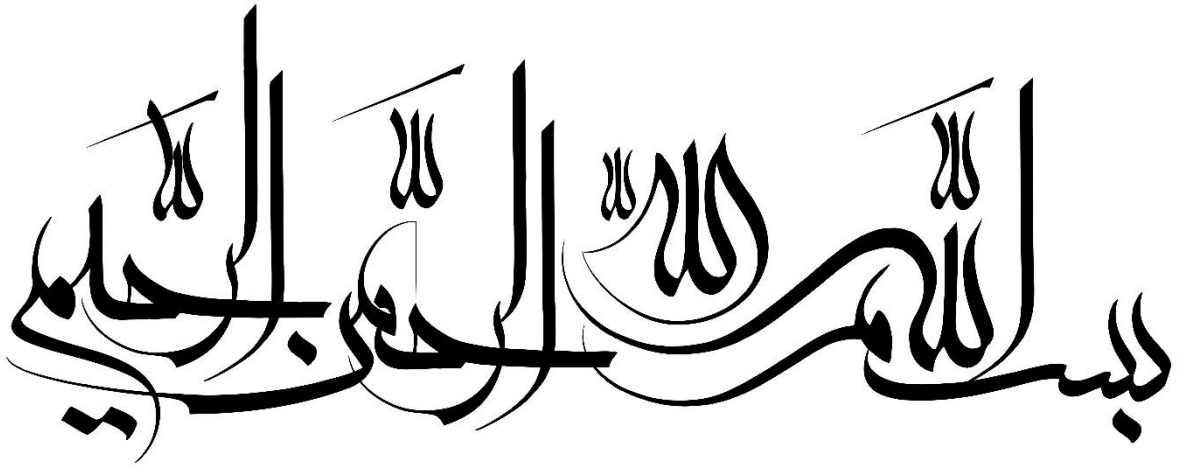

عنوان : تفاوت‌های اختصاص دادن حافظه در اِستَک و هیپ

استاد: میثاق یاریان

نام دانشجو : سید محمد موسوی مطلق

درس: برنامه نویسی سمت سرور

نیمسال : اول ۱۴۰۲

تفاوت‌های اختصاص دادن حافظه در اِستَک و هیپ

با توجه به اینکه، از اوایل سیستم‌های کامپیوتری این تمایز در این وجود داشته است که برنامه های اصلی در حافظه فقط خواندنی مانند **ROM**، **PROM** و یا **EEPROM**نگه داری می‌شوند. به عنوان دیگر از زمانی که سیستم ها پیچیده‌تر شدند برنامه‌ها از حافظه‌های دیگری مانند **RAM**به جای اجرا در حافظه **ROM**استفاده کردند. این ایده به خاطر این بود که تعدادی از قسمت های حافظه مربوط به برنامه نباید تغییر یابند و در این حالت باید حفظ شوند. در این میان دو بخش `.text` و `.rodata`بخشی‌هایی از برنامه هستند که می‌تواند به بخش های دیگر برای وظایف خاص تقسیم شوند که در ادامه به آن‌ها اشاره شده است .

- بخش کد، به عنوان یک بخش متنی (`.text`) و یا به طور ساده به عنوان متن شناخته می‌شود. جایی است که بخشی از یک فایل شیء یا بخش مربوطه از فضای آدرس مجازی برنامه که حاوی دستورالعمل های اجرایی است و به طور کلی فقط خواندنی بوده و اندازه ثابتی دارد می‌باشد .
- بخش `.bss`که به عنوانی بخشی ویژه (محل نگه داری اطلاعات تخصیص داده نشده (مقدار دهی نشده)) محلی که متغیرهای سراسری و ثابت با مقدار صفر شروع می‌شوند .
- بخش داده (`.data`) حاوی هر گونه متغیر سراسری و یا استاتیک که دارای یک مقدار از پیش تعریف شده هستند و می‌توانند اصلاح شوند .

کد زیر طرح معمولی از یک حافظه برنامه ساده کامپیوتری را با متن، داده های مختلف، و بخش‌های استک و هیپ و bss را نشان می‌دهد .

```
.section.data

< initialized data here>

.section .bss

< uninitialized data here>

.section .text

.globl _start

_start:

<instruction code goes here>
```

برای مثال در کد C به صورت زیر خواهد بود :

```
int val = 3;

char string[] = "Hello World";
```

مقادیر برای این نوع متغیرها در ابتدا در حافظه فقط خواندنی ذخیره می‌شوند. (معمولا در داخل .text) و در زمان اجرای برنامه که به صورت روتین خواهد بود در بخش .data کپی می‌شوند .

بخش BSS یا همانBSS.

در برنامه‌نویسی کامپیوتر، نام .bss یا bssتوسط بسیاری از کامپایلرها و لینکرها برای بخشی از دیتا سِگمنت (Data Segment) استفاده می‌شود که حاوی متغیر های استاتیک اختصاصی که تنها از بیت هایی با ارزش صفر شروع شده است می‌باشد. این بخش به عنوان **BSS Section**و یا **BSS Segment**شناخته می‌شود .

به طور معمول فقط طول بخش **bss**نه **data**در فایل آبجکت ذخیره می‌شود. برای نمونه، یک متغیر به عنوان استاتیک تعریف شده است static int i;این در بخش BSS خواهد بود .

حافظه هیپ (Heap)

ناحیه هیپ (Heap) به طور رایج در ابتدای بخش‌های .bss و .data قرار گرفته است و به اندازه‌های آدرس بزرگتر قابل رشد است. ناحیه هیپ توسط توابع malloc, calloc, realloc و free مدیریت می‌شود که ممکن است توسط سیستم‌های brk و sbrkجهت تنظیم اندازه مورد استفاده قرار گیرد. ناحیه هیپ توسط تمامی نخ‌ها، کتابخانه‌های مشترک و ماژول‌های بارگذاری شده در یک فرآیند به اشتراک گذاشته می‌شود .

به طور کلی حافظه Heap بخشی از حافظه کامپیوتر شما است که به صورت خودکار برای شما مدیریت نمی‌شود، و به صورت محکم و مطمئن توسط پردازنده مرکزی مدیریت نمی‌شود. آن

بیشتر به عنوان یک ناحیه شناور بسیار بزرگی از حافظه است. برای اختصاص دادن حافظه در ناحیه هیپ شما باید از توابع malloc(), calloc()که توابعی از C هستند استفاده کنید. یکبار که شما حافظه ای را در ناحیه هیپ اختصاص دهید، جهت آزاد سازی آن باید خود مسئول باشید و با استفاده از تابع free() این کار را به صورت دستی جهت آزاد سازی حافظه اختصاص یافته شده انجام دهید. اگر شما در این کار موفق نباشید، برنامه شما در وضعیت نَشت حافظه (Memory Leak) قرار خواهد گرفت. این بدین معنی است که حافظه اختصاص یافته شده در هیپ هنوز خارج از دسترس قرار گرفته و مورد استفاده قرار نخواهد گرفت. این وضعیت همانند گرفتگی رَگ در بدن انسان است و حافظه نشت شده جهت عملیات در دسترس نخواهد بود. خوشبختانه ابزارهایی برای کمک کردن به شما در این زمینه موجود هستند که یکی از آن‌ها Valgrind نام دارد و شما می‌توانید در زمان اشکال زدائی از آن جهت تشخیص نواحی نشت دهنده حافظه استفاده کنید .

بر خلاف حافظه اِستک (Stack) حافظه هیپ محدودیتی در اندازه متغیرها ندارد)جدا از محدودیت آشکار فیزیکی در کامپیوتر شما). حافظه هیپ در خواندن کمی کُند تر از نوشتن نسبت به حافظه اِستک است، زیرا جهت دسترسی به آن‌ها در حافظه هیپ باید از اشاره گر استفاده شود. بر خلاف حافظه اِستک، متغیرهایی که در حافظه هیپ ساخته می‌شوند توسط هر تابعی در هر بخشی از برنامه شما در دسترس بوده و اساسا متغیرهای تعریف شده در هیپ در دامنه سراسری قرار دارند .

حافظه اِستک (Stack)

ناحیه اِستک (Stack) شامل برنامه اِستک، با ساختار **LIFO**کوتاه شده عبارت **Last In First Out** (آخرین ورودی از همه زودتر خارج می‌شود) به طور رایج در بالاترین بخش از حافظه قرار می‌گیرد. یک (اشاره گر پشته) در بالاترین قسمت اِستک قرار می‌گیرد. زمانی که تابعی فراخوانی می‌شود این تابع به همراه تمامی متغیرهای محلی خودش در داخل حافظه اِستک قرار می‌گیرد و با فراخوانی یک تابع جدید تابع جاری بر روی تابع قبلی قرار می‌گیرد و کار به همین صورت درباره دیگر توابع ادامه پیدا می‌کند .

مزیت استفاده از حافظه اِستک در ذخیره متغیرها است، چرا که حافظه به صورت خودکار برای شما مدیریت می‌شود. شما نیازی برای اختصاص دادن حافظه به صورت دستی ندارید، یا نیازی به آزاد سازی حافظه ندارید. به طور کلی دلیل آن نیز این است که حافظه اِستک به اندازه کافی توسط پردازنده مرکزی بهینه و سازماندهی می‌شود. بنابراین خواندن و نوشتن در حافظه اِستک بسیار سریع است .

کلید درک حافظه اِستک در این است که زمانی که تابع خارج می‌شود، تمامی متغیرهای موجود در آن همراه با آن خارج و به پایان زندگی خود میرسند .بنابراین متغیرهای موجود در حافظه اِستک به طور طبیعی به صورت محلی هستند .این مرتبط با مفهوم دامنه متغیرها است که قبلا از آن یاد شده است، یا همان متغیرهای محلی در مقابل متغیرهای سراسری .

یک اشکال رایج در برنامه نویسی C تلاش برای دسترسی به یک متغیر که در حافظه اِستک برای یک تابع درونی ساخته شده است می‌باشد. یعنی از یک مکان در برنامه شما به خارج از تابع (یعنی زمانی که آن تابع خارج شده باشد) رجوع می‌کند .

یکی دیگر از ویژگی‌های حافظه اِستک که بهتر است به یاد داشته باشید این است که، محدودیت اندازه (نسبت به نوع سیستم عامل متفاوت) است. این مورد در حافظه هیپ صدق نمی‌کند .

خلاصه ای از حافظه اِستک (Stack)

- حافظه اِستک متناسب با ورود و خروج توابع و متغیرهای درونی آن‌ها افزایش و کاهش می‌یابد
- نیازی برای مدیریت دستی حافظه برای شما وجود ندارد، حافظه به طور خودکار برای متغیرها اختصاص و در زمان نیاز به صورت خودکر آزاد می‌شود
- در اِستک اندازه محدود است
- متغیرهای اِستک تنها در زمان اجرای تابع ساخته می‌شوند
- مزایا و معایب حافظه اِستک و هیپ

حافظه اِستک (Stack)

- دسترسی بسیار سریع به متغیرها
- نیازی برای باز پس گیری حافظه اختصاص یافته شده ندارید
- فضا در زمان مورد نیاز به اندازه کافی توسط پردازنده مرکزی مدیریت می‌شود، حافظه ای نشت نخواهد کرد
- متغیرها فقط محلی هستند
- محدودیت در حافظه اِستک بسته به نوع سیستم عامل متفاوت است
- متغیرها نمی‌توانند تغییر اندازه دهند

حافظه هیپ (Heap)

- متغیرها به صورت سراسری قابل دسترس هستند
- محدودیتی در اندازه حافظه وجود ندارد
- تضمینی برای حافظه مصرفی وجود ندارد، ممکن است حافظه در زمان‌های خاص از برنامه نشت کرده و حافظه اختصاص یافته شده برای استفاده در عملیات دیگر آزاد نخواهد شد
- شما باید حافظه را مدیریت کنید، شما باید مسئولیت آزاد سازی حافظه های اختصاص یافته شده به متغیرها را بر عهده بگیرید
- اندازه متغیرها می‌تواند توسط تابع `realloc()` تغییر یابد

در اینجا یک برنامه کوتاه وجود دارد که در آن متغیرها در یک حافظه اِستک ایجاد شده اند .

```
#include <stdio.h>



double multiplyByTwo (double input) {



double twice = input * 2.0;
```

```
return twice;




}



int main (int argc, char *argv[])

{



int age = 30;

double salary = 12345.67;

double myList[3] = {1.2, 2.3, 3.4};
```

```
printf("double your salary is %.3f\n", multiplyByTwo(salary));



return 0;



}
```

ما متغیرهایی را اعلان کرده‌ایم که یک int، یک doubleو یک آرایه که سه نوع doubleدارد هستند. این متغیرها داخل اِستک وارد و به زودی توسط تابع main در زمان اجرا حافظه مورد نیاز خود را دریافت خواهند کرد. زمانی که تابع main خارج می‌شود (برنامه متوقف می‌شود) این متغیرها همگی از داخل حافظه اِستک خارج خواهند شد. به طور مشابه، در تابع multiplByTwo()متغیر twiceکه از نوع doubleاست، داخل اِستک وارد شده و در زمان اجرای تابع multiplyByTwo()حافظه به آن اختصاص می‌یابد. زمانی که تابع فوق خارج شود یعنی به نقطه پایان اجرایی خود برسد، حافظه اختصاص یافته شده به متغیرهای داخلی آن نیز آزاد خواهند شد. به طور خلاصه توجه داشته باشید که تمامی متغیرهای محلی در این نوع تعریف تنها در طول زمان اجرایی زمانی تابع زنده هستند .

به عنوان یک یادداشت جانبی، روشی برای نگه داری متغیرها در حافظه اِستک وجود دارد، حتی در زمانی که تابع خارج می‌شود. آن روش توسط کلمه کلیدی staticممکن خواهد شد که در زمان اعلان متغیر استفاده می‌شود. متغیری که توسط کلمه کلیدی staticتعریف

می‌شود، بنابراین چیزی مانند متغیر از نوع سراسری خواهد بود، اما تنها در داخل تابعی که داخل آن ایجاد شده است قابل مشاهده خواهد بود. این یک ساختار عجیب و غریب است، که احتمالا به جز شرایط بسیار خاص نیازی به آن نباشد .

نسخه دیگری از برنامه فوق در قالب حافظه هیپ به صورت زیر است :

```
#include <stdio.h>

#include <stdlib.h>



double *multiplyByTwo (double *input) {



double *twice = malloc(sizeof(double));

*twice = *input * 2.0;

return twice;
```

```
}



int main (int argc, char *argv[])

{

int *age = malloc(sizeof(int));

*age = 30;

double *salary = malloc(sizeof(double));

*salary = 12345.67;

double *myList = malloc(3 * sizeof(double));
```

```
myList[0] = 1.2;

myList[1] = 2.3;

myList[2] = 3.4;



double *twiceSalary = multiplyByTwo(salary);



printf("double your salary is %.3f\n", *twiceSalary);



free(age);

free(salary);

free(myList);
```

```
free(twiceSalary);



return 0;

}
```

همانطور که می‌بینید، استفاده از malloc() برای تخصیص حافظه در حافظه Heap و استفاده از free() جهت آزاد سازی حافظه تخصیص یافته می‌باشد. این مواجه شدن چیز بسیار بزرگی محسوب نمی‌شود اما کمی مبهم است. چیز دیگری که باید به آن توجه داشته باشید علامت ستاره (*) است که در همه جای کدها دیده می‌شود. اینها چه چیزهایی هستند؟ پاسخ این سوال این است : اینها اشاره گر هستند !

توابع malloc() و calloc() و free() با اشاره‌گرهایی مواجه می‌شوند که مقادیرشان واقعی نیست. اشاره گرها نوع داده ای خاصی در C هستند که آدرس حافظه مربوطه را بر می‌گردانند. در خط ۵ متغیر twice یک متغیر از نوع double نیست، اما اشاره به یک double دارد. آن آدرس حافظه ای است که نوع double در آن بلوک از حافظه ذخیره شده است .

در C++ توسط کلمه کلیدی new که خود آن نیز یک اپراتور محسوب می‌شود می‌توان حافظه ای را در Heap اختصاص داد. به عنوان مثال :

```
int* myInt = new int(256);
```

آدرس‌های موجود در حافظه توسط اپراتور new به اشاره‌گر مربوطه پاس داده می‌شود. به مثال زیر توجه کنید، متغیر تعریف شده در حافظه اِستک قرار گرفته است :

```
int variable = 256;
```

سوالی که ممکن است افراد کنجکاو از خود بپرسند این است که چه زمانی از Stack و چه زمانی از Heap باید استفاده کنیم؟ !

خب پاسخ این سوال اینگونه خواهد بود، زمانی که شما نیاز به یک بلوک بسیار بزرگی از حافظه دارید، که در آن یک ساختار بزرگ یا یک ارایه بزرگی را ذخیره کنید و نیاز داشته باشید که متغیرهای شما به مدت طولانی در سرتاسر برنامه شما در دسترس باشند در این صورت از حافظه Heap استفاده کنید .

در صورتی که شما نیاز به متغیرهای کوچکی دارید که تنها نیاز است در زمان اجرای تابع در دسترس باشند و قابلیت خواندن و نوشتن سریعتری داشته باشند از نوع حافظه Stack استفاده کنید. فقط فراموش نکنید که حافظه Heap تحت توابع `molloc()`, `realloc()`, `calloc()`و `free()` مدیریت می‌شوند. هرچند اشاره‌گر های هوشمند نیز در C++ وجود دارند اما در بسیاری از مواقع که نیاز است بسیار جزئی و حساس بر روی کدهای خود کار کنید از مدیریت حافظه به صورت دستی استفاده کنید .