# آزمایش ۶ - مدیریت حافظه

## ۶.۱ مقدمه

در این جلسه از آزمایشگاه با ساختار حافظه ی پردازه ها آشنا خواهیم شد.

## ۶.۲ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

1. برنامه‌نویسی به زبان C++/C
2. دستورات پوسته‌ی لینوکس که در جلسات قبل فرا گرفته شده‌اند.

## ۶.۳ مدیریت حافظه

همان طور که می دانید، در زبان های برنامه‌نویسی سطح بالا، مانند C، هنگامی که یک متغیر تعریف می کنید، کامپایلر فضای مورد نیاز برای آن را در نظر می گیرد و نیازی به تخصیص فضا به صورت دستی ندارید. متغیر های سراسری در Data Segment پردازه و متغیر های محلی در Stack Segment قرار می گیرند.

همچنین در برخی از شرایط نیاز است که حافظه به صورت پویا اختصاص یابد؛ برای مثال برای ایجاد یک داده‌ساختار مانند درخت یا لیست پیوندی. در زبان برنامه نویسی C، توابع استاندارد malloc و free برای این منظور وجود دارند. این توابع با استفاده از فراخوانی های سیستمی، عمل مدیریت حافظه به صورت پویا را انجام می‌دهند.

## ۶.۴ شرح آزمایش

(آ) استفاده از توابع malloc و free

- ساختار زیر را در نظر بگیرید:

```
struct MyStruct {
    int a;
    int b;
    char name[20];
};
```

- به کمک malloc حافظه ی مورد نیاز برای یک instance از این ساختار را تخصیص دهید. خروجی دستور malloc چیست؟
- برای فیلد های instance ایجاد شده مقادیری را اختصاص دهید و آن ها را چاپ کنید.
- به کمک free حافظه ی گرفته شده را آزاد کنید.

(ب) مشاهده ی وضعیت حافظه ی پردازه ها

- به کمک دستور زیر وضعیت حافظه ی پردازه ها را مشاهده کنید:

```
ps -o user,vsz,rss,pmem,fname -e
```

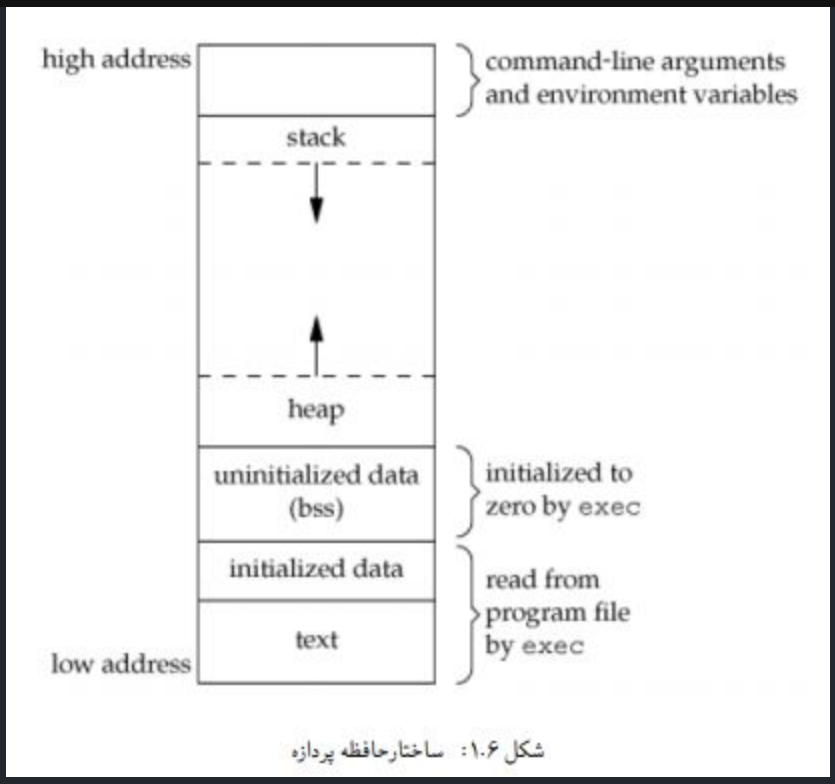
- به کمک دستور `man ps` توضیح دهید که هر کدام از ستون‌ها چه اطلاعاتی را نشان می دهند.

ج) اجزای حافظه ی یک پردازه

حافظه ی یک پردازه در معماری های امروزی به اجزای زیر تقسیم بندی می شود:

- بخش text که کد زبان ماشین پردازه را نگه داری می کند؛
- بخش data که متغیر های سراسری پردازه در آن قرار می گیرد، برخی از متغیر ها دارای مقدار اولیه هستند و برخی خیر. دسته ی دوم در بخشی به نام bss واقع می شوند؛
- بخش heap که شامل حافظه هایی است که به صورت پویا اختصاص داده می شوند؛
- بخش stack که متغیر های محلی، پارامتر های توابع و مقادیر بازگشتی آن ها در آن قرار دارند.

تصویر زیر این ساختار را نمایش می دهد (آدرس های کوچک تر در پایین تصویر قرار دارند):




شکل ۱.۶: ساختارحافظه پردازه

- به کمک دستور which محل قرار گیری دستور ls را درون فایل‌سیستم پیدا کنید.
- با استفاده از دستور size مشاهده کنید که چه مقدار از کد ماشین این دستور به بخش هایی که در بالا اشاره شد اختصاص دارد. این دستور کدام یک از بخش های بالا را نشان **نمی دهد**؟

د) اشتراک حافظه

در سیستم‌عامل لینوکس، معماری حافظه به صورت صفحه‌بندی شده است؛ به این معنا که حافظه در تکه هایی (معمولا ۸۱۹۲ یا ۴۰۹۶ بایتی) به پردازه ها اختصاص می یابد. سیستم‌عامل می تواند بعضی از این تکه ها را بین پردازه های مختلف به اشتراک بگذارد. برای مثال تمام پردازه هایی که یک کد یکسان را اجرا می کنند، بخش text مشترک دارند. کاربرد دیگر اشتراک این صفحات برای کتابخانه های مشترک است. برای مثال، اکثر برنامه ها از تابع printf استفاده می کنند؛ بنابراین منطقی است که تنها یکبار آن را در حافظه آورده و بین تمام پردازه هایی که از آن استفاده می کنند، حافظه را به اشتراک بگذاریم.

- به کمک دستور ldd کتابخانه های مشترکی که توسط دستور ls استفاده شده است را مشاهده کنید.
- این کار را برای چند برنامه ی دیگر (برای مثال nano) امتحان کنید و نتیجه را در گزارش کار خود بیاورید.

ه) آدرس های بخش های مختلف حافظه ی پردازه

به کمک استفاده از نماد (symbol) های خاصی که در لینوکس تعریف شده اند، قادر هستیم که آدرس پایان segment code ، آدرس پایان segment global و همچنین آدرس پایان بخش متغیر های سراسری دارای مقدار اولیه را مشاهده کنیم.

برای این کار، سه نماد زیر در دسترس هستند:

- etext : در این نماد قرار دارد text اولین آدرس بعد از پایان.
- edata: اولین آدرس بعد از پایان متغیر های سراسری دارای مقدار اولیه در این نماد قرار دارد.
- end: در این نماد قرار می گیرد bss اولین آدرس بعد از پایان بخش.

در ادامه موارد زیر را انجام دهید:

- به کمک دستور `man etext` ، جزئیات بیشتری در مورد نحوه ی استفاده از این نماد ها ببینید و کدی که به عنوان مثال در ادامه این راهنما آمده است را نوشته و اجرا کنید.
- آیا خروجی این برنامه با ساختاری که در تصویر قبل آمده است تطابق دارد؟

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

extern char etext, edata, end; /* The symbols must have some type,
                                  or "gcc -Wall" complains */

int
main(int argc, char *argv[])
{
    printf("First address past:\n");
    printf("    program text (etext)      %10p\n", &etext);
    printf("    initialized data (edata)  %10p\n", &edata);
```



- پس از مطالعه ی این لینک، توضیح دهید که معنای کامنت زیر در کدی که در بالا ذکر شده چیست. چرا به جای «متغیر»، به etext و edata و end، «نماد» گفته می‌شود؟ چرا در تعریف آنها از extern استفاده می‌شود؟

```c
extern char etext, edata, end; /* The symbols must have some type,
                                  or "gcc -Wall" complains */
```

- برای مشاهده ی آدرس انتهای heap از فراخوانی سیستمی sbrk با مقدار آرگومان 0 استفاده می شود. حال برنامه ای بنویسید که ابتدا، آدرس انتهای heap را چاپ کند. سپس در یک حلقه، یک مقدار ثابت نسبتا کم (مثلاً اکیلوبایت) از حافظه را به کمک malloc به خود اختصاص دهد، تا زمانی که آدرس انتهای heap مقداری متفاوت از هنگام شروع برنامه پیدا

کند. و بعد تعداد تکرار های حلقه را چاپ کند. انتظار می‌رود که پس از یک بار malloc کردن، آدرس انتهای heap تغییر کند و لذا تعداد تکرار های حلقه، برابر عدد ۱ باشد. امّا مشاهده می‌شود که اینگونه نیست! درباره ی دلیل این خروجی توضیح دهید.

- همان طور که گفته شد stack در جهت عکس heap رشد می کند. همچنین، متغیر های محلی و آرگومان های توابع در stack قرار می گیرند. یک تابع بازگشتی بنویسید که ابتدا یک متغیر محلی i ایجاد کند؛ سپس آدرس i را چاپ کرده و دوباره خودش را فراخوانی کند. روند تغییر آدرس متغیر i چگونه است؟ (برنامه را به گونه ای محدود کنید تا عملیات بازگشت مثلا ۱۰۰ بار انجام شود).

## مطالعه ی بیشتر

پیشنهاد می‌شود برای مطالعه ی بیشتر در مورد حافظه ی پردازه ها، به لینک زیر مراجعه کنید.

- https://blog.holbertonschool.com/hack-the-virtual-memory-malloc-the-heap-the-program-break/