



گزارش آزمایش ۶: مدیریت حافظه

ملیکا علیزاده ۴۰۱۱۰۶۲۵۵

الینا هژبری ۴۰۱۱۷۰۶۶۱

آ استفاده از فراخوانی‌های سیستمی malloc و free

در این بخش و با کد زیر ابتدا با استفاده از malloc تخصیص حافظه برای یک نمونه از MyStruct انجام و به فیلدهای آن مقداردهی شده است و در نهایت آدرس و مقادیر آن قبل و بعد از استفاده از فراخوانی سیستمی free چاپ شده است. خروجی تابع malloc یک آدرس (pointer) از نوع void* است که نشان دهنده‌ی ابتدای حافظه‌ی اختصاص داده شده در heap است.

```
GNU nano 7.2 malloc.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

int main() {

    struct MyStruct *ptr = (struct MyStruct *) malloc(sizeof(struct MyStruct));

    ptr->a = 10;
    ptr->b = 20;
    strcpy(ptr->name, "melika");

    printf("--before using free--\n");
    printf("MyStruct address = %p\n", ptr);
    printf("a = %d\n", ptr->a);
    printf("b = %d\n", ptr->b);
    printf("name = %s\n", ptr->name);

    free(ptr);

    printf("--after using free--\n");
    printf("MyStruct address = %p\n", ptr);
    printf("a = %d\n", ptr->a);
    printf("b = %d\n", ptr->b);
    printf("name = %s\n", ptr->name);

    return 0;
}
```

شکل ۱: کد استفاده از malloc و free

```
401106255-401170661@ubuntu: ~/oslab6
401106255-401170661@ubuntu:~/oslab6$ nano malloc.c
401106255-401170661@ubuntu:~/oslab6$ gcc malloc.c -o malloc
401106255-401170661@ubuntu:~/oslab6$ ./malloc
--before using free--
MyStruct address = 0x5a7d196902a0
a = 10
b = 20
name = melika
--after using free--
MyStruct address = 0x5a7d196902a0
a = -1479436656
b = 5
name = C+|
```

شکل ۲: نمایش خروجی کد

ب مشاهدهی وضعیت حافظه‌ی پردازها

```
401106255-401170661@ubuntu: ~/oslab6
401106255-401170661@ubuntu:~/oslab6$ ps -o user,vsz,rss,pmem,fname -e
USER      VSZ    RSS %MEM COMMAND
root      23100 13780 0.1 systemd
root         0     0 0.0 kthreadd
root         0     0 0.0 pool_wor
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 rcu_task
root         0     0 0.0 rcu_task
root         0     0 0.0 rcu_task
root         0     0 0.0 ksoftirq
root         0     0 0.0 rcu_pree
root         0     0 0.0 rcu_exp_
root         0     0 0.0 rcu_exp_
root         0     0 0.0 migratio
4011062+ 3080296 66748 0.8 gjs
4011062+ 701632 14252 0.1 xdg-desk
4011062+ 1011656 87368 1.0 xdg-desk
4011062+ 236272 6336 0.0 gvfsd-me
4011062+ 417236 24764 0.3 xdg-desk
4011062+ 494460 30464 0.3 update-n
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root      592532 41672 0.5 fwupd
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
root         0     0 0.0 kworker/
4011062+ 11560812 451660 5.5 firefox
4011062+ 21284 4192 0.0 crashhel
4011062+ 295816 33828 0.4 forkserve
```

شکل ۳: خروجی دستور `ps -o user,vsz,rss,pmem,fname -e`

- `user`: نام کاربری که مالک پردازه است.
- `vsz`: مقدار حافظه مجازی به کیلوبایت که توسط پردازه استفاده می‌شود. شامل کد، داده، `heap` و `stack`.
- `rss`: اندازه‌ی حافظه‌ی فیزیکی به کیلوبایت در `RAM` که توسط پردازه استفاده می‌شود.
- `pmem`: درصدی از حافظه فیزیکی کل که پردازه مشغول کرده است.
- `fname`: ۸ بایت اول از نام فایل اجرایی پردازه.

ج اجزای حافظه‌ی یک پردازنده

```
401106255-401170661@ubuntu: ~/oslab6
401106255-401170661@ubuntu:~/oslab6$ which ls
/usr/bin/ls
```

شکل ۴: دریافت محل قرارگیری دستور ls با استفاده از دستور which

```
401106255-401170661@ubuntu:~/oslab6$ size /usr/bin/ls
  text    data     bss     dec     hex filename
127793    4936    4760 137489   21911 /usr/bin/ls
```

شکل ۵: مقادیری که کد ماشین دستور ls به هر بخش از ساختار حافظه‌ی پردازنده اختصاص داده‌است. (با استفاده از دستور size)

- در دستور size بخش‌های heap و stack وجود ندارند زیرا در زمان اجرای پردازنده و توسط سیستم‌عامل مدیریت می‌شوند.

```
401106255-401170661@ubuntu:~/oslab6$ ldd /bin/ls
linux-vdso.so.1 (0x000073a16ff34000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x000073a16fec0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000073a16fc00000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x000073a16fe33000)
/lib64/ld-linux-x86-64.so.2 (0x000073a16ff36000)
```

شکل ۶: نمایش کتابخانه‌های مشترکی که توسط دستور ls استفاده شده‌است. (با استفاده از دستور ldd)

```
401106255-401170661@ubuntu:~/oslab6$ ldd /bin/nano
linux-vdso.so.1 (0x00007d8e61374000)
libncursesw.so.6 => /lib/x86_64-linux-gnu/libncursesw.so.6 (0x00007d8e612de000)
libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007d8e612aa000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007d8e61000000)
/lib64/ld-linux-x86-64.so.2 (0x00007d8e61376000)
```

شکل ۷: نمایش کتابخانه‌های مشترکی که توسط دستور nano استفاده شده‌است.

```
401106255-401170661@ubuntu:~/oslab6$ ldd /bin/bash
linux-vdso.so.1 (0x000071b7e1068000)
libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x000071b7e0eb3000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000071b7e0c00000)
/lib64/ld-linux-x86-64.so.2 (0x000071b7e106a000)
401106255-401170661@ubuntu:~/oslab6$
```

شکل ۸: نمایش کتابخانه‌های مشترکی که توسط دستور bash استفاده شده‌است.

```
401106255-401170661@ubuntu:~/oslab6$ ldd /bin/gcc
linux-vdso.so.1 (0x00007d92053b3000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007d9205000000)
/lib64/ld-linux-x86-64.so.2 (0x00007d92053b5000)
401106255-401170661@ubuntu:~/oslab6$
```

شکل ۹: نمایش کتابخانه‌های مشترکی که توسط دستور gcc استفاده شده‌است.

ه آدرس‌های بخش‌های مختلف حافظه‌ی پردازنده

اجرای کد برای استفاده از متغیرهای `extern`

```
GNU nano 7.2 test.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

extern char etext, edata, end;

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);
    printf("    uninitialized data (end)    %10p\n", &end);

    return 0;
}
```

شکل ۱۰: کد استفاده از متغیرهای `extern`

```
401106255-401170661@ubuntu:~/oslab6$ nano test.c
401106255-401170661@ubuntu:~/oslab6$ gcc test.c -o test
401106255-401170661@ubuntu:~/oslab6$ ./test
First address past:
    program text (etext)    0x640aeed4b1f9
    initialized data (edata)    0x640aeed4e010
    uninitialized data (end)    0x640aeed4e018
401106255-401170661@ubuntu:~/oslab6$
```

شکل ۱۱: خروجی کد

همانطور که در خروجی کد مشخص است، آدرس‌ها طبق ساختاری که در تصویر آمده‌است هستند یعنی پایین‌ترین آدرس مربوط به `text` و در متغیر `etext` و در وسط آدرس پایان متغیرها یعنی `edata` و شروع `heap` در متغیر `end` در بالاترین آدرس است.

```

GNU nano 7.2                                heap.c *
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    void *initial_brk = sbrk(0);
    printf("Initial end of heap address: %p\n", initial_brk);

    void *p1 = malloc(100);
    void *after_malloc = sbrk(0);
    printf("After malloc(100): %p\n", after_malloc);

    int difference = after_malloc - initial_brk;
    printf("difference: %d\n", difference);

    return 0;
}

```

شکل ۱۲: استفاده از فراخوانی سیستمی sbrk برای مشاهده‌ی آدرس انتهای heap بعد از اجرای malloc

```

401106255-401170661@ubuntu:~/oslab6$ nano heap.c
401106255-401170661@ubuntu:~/oslab6$ gcc heap.c -o heap
401106255-401170661@ubuntu:~/oslab6$ ./heap
Initial end of heap address: 0x5e929d682000
After malloc(100): 0x5e929d6a3000
difference: 135168
401106255-401170661@ubuntu:~/oslab6$

```

شکل ۱۳: خروجی کد

```
GNU nano 7.2                                stack.c
#include <stdio.h>

void test_stack(int num) {
    int i;
    printf("Address of i: %p\n", &i);
    if (num < 100) {
        test_stack(num + 1);
    }
}

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

شکل ۱۴: کد تابع بازگشتی


```
401106255-401170661@ubuntu:~/oslab6$ nano stack.c
401106255-401170661@ubuntu:~/oslab6$ gcc stack.c -o stack
401106255-401170661@ubuntu:~/oslab6$ ./stack
Address of i: 0x7ffd0d770104
Address of i: 0x7ffd0d7700d4
Address of i: 0x7ffd0d7700a4
Address of i: 0x7ffd0d770074
Address of i: 0x7ffd0d770044
Address of i: 0x7ffd0d770014
Address of i: 0x7ffd0d76ffe4
Address of i: 0x7ffd0d76ffb4
Address of i: 0x7ffd0d76ff84
Address of i: 0x7ffd0d76ff54
Address of i: 0x7ffd0d76ff24
Address of i: 0x7ffd0d76fef4
Address of i: 0x7ffd0d76fec4
Address of i: 0x7ffd0d76fe94
Address of i: 0x7ffd0d76fe64
Address of i: 0x7ffd0d76fe34
Address of i: 0x7ffd0d76fe04
Address of i: 0x7ffd0d76fdd4
Address of i: 0x7ffd0d76fda4
Address of i: 0x7ffd0d76fd74
Address of i: 0x7ffd0d76fd44
Address of i: 0x7ffd0d76fd14
Address of i: 0x7ffd0d76fce4
Address of i: 0x7ffd0d76fcb4
Address of i: 0x7ffd0d76fc84
Address of i: 0x7ffd0d76fc54
Address of i: 0x7ffd0d76fc24
Address of i: 0x7ffd0d76fbf4
Address of i: 0x7ffd0d76fbc4
Address of i: 0x7ffd0d76fb94
Address of i: 0x7ffd0d76fb64
Address of i: 0x7ffd0d76fb34
```

شکل ۱۵: خروجی کد