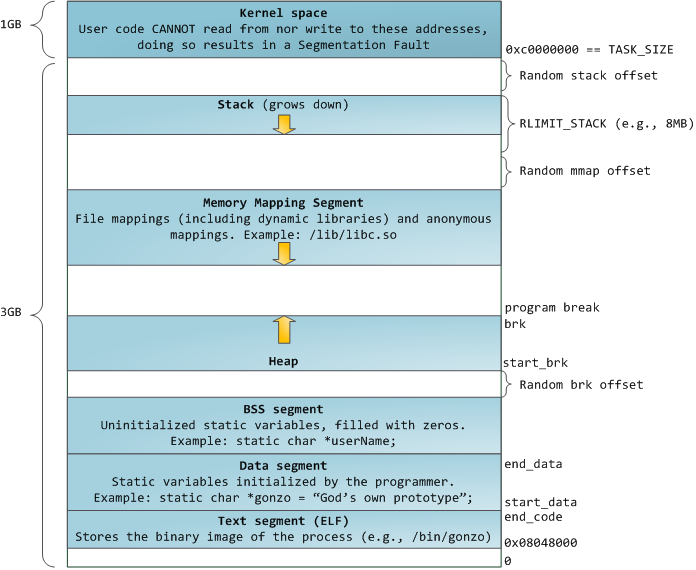
**Syscalls used by malloc.**

Posted on [February 11, 2015](https://sploitfun.wordpress.com/2015/02/11/syscalls-used-by-malloc/) by [sploitfun](https://sploitfun.wordpress.com/author/sploitfun/)

Having landed on this page, you should know malloc uses syscalls to obtain memory from the OS. As shown in the below picture malloc invokes either [brk](http://man7.org/linux/man-pages/man2/sbrk.2.html) or [mmap](http://man7.org/linux/man-pages/man2/mmap.2.html) syscall to obtain memory.

**brk**: [brk](http://lxr.free-electrons.com/source/mm/mmap.c?v=3.8#L252) obtains memory (non zero initialized) from kernel by increasing program break location ([brk](http://lxr.free-electrons.com/source/include/linux/mm_types.h?v=3.8#L365)). Initially start ([start\_brk](http://lxr.free-electrons.com/source/include/linux/mm_types.h?v=3.8#L365)) and end of heap segment ([brk](http://lxr.free-electrons.com/source/include/linux/mm_types.h?v=3.8#L365)) would point to same location.

* When [ASLR](http://en.wikipedia.org/wiki/Address_space_layout_randomization) is turned off, start\_brk and brk would point to end of data/bss segment ([end\_data](http://lxr.free-electrons.com/source/include/linux/mm_types.h?v=3.8#L364)).
* When ASLR is turned on, start\_brk and brk would be equal to end of data/bss segment (end\_data) plus random brk offset.

[](http://static.duartes.org/img/blogPosts/linuxFlexibleAddressSpaceLayout.png)

Above “process virtual memory layout” picture shows start\_brk is the beginning of heap segment and brk (program break) is the end of heap segment.

*Example*:

1. */\* sbrk and brk example \*/*
2. #include <stdio.h>
3. #include <unistd.h>
4. #include <sys/types.h>
6. int main()
7. {
8. void \*curr\_brk, \*tmp\_brk = NULL;
10. printf("Welcome to sbrk example:%d\n", getpid());
12. */\* sbrk(0) gives current program break location \*/*
13. tmp\_brk = curr\_brk = sbrk(0);
14. printf("Program Break Location1:%p\n", curr\_brk);
15. getchar();
17. */\* brk(addr) increments/decrements program break location \*/*
18. brk(curr\_brk+4096);
20. curr\_brk = sbrk(0);
21. printf("Program break Location2:%p\n", curr\_brk);
22. getchar();
24. brk(tmp\_brk);
26. curr\_brk = sbrk(0);
27. printf("Program Break Location3:%p\n", curr\_brk);
28. getchar();
30. return 0;
31. }

*Output Analysis*:

*Before increasing program break*: In the below output we can observe there is NO heap segment. Hence

* start\_brk = brk = end\_data = 0x804b000.

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ ./sbrk

Welcome to sbrk example:6141

Program Break Location1:**0x804b000**

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6141/maps

...

**0804a000-0804b000 rw-p 00001000 08:01 539624 /home/sploitfun/ptmalloc.ppt/syscalls/sbrk**

b7e21000-b7e22000 rw-p 00000000 00:00 0

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$

*After increasing program break location*: In the below output we can observe there is heap segment. Hence

* start\_brk = end\_data = 0x804b000
* brk = 0x804c000.

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ ./sbrk

Welcome to sbrk example:6141

Program Break Location1:0x804b000

Program Break Location2:**0x804c000**

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6141/maps

...

0804a000-0804b000 rw-p 00001000 08:01 539624 /home/sploitfun/ptmalloc.ppt/syscalls/sbrk

**0804b000-0804c000 rw-p 00000000 00:00 0 [heap]**

b7e21000-b7e22000 rw-p 00000000 00:00 0

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$

where  
*0804b000-0804c000* is Virtual address range for this segment  
*rw-p* is Flags (Read, Write, NoeXecute, Private)  
*00000000* is File offset – Since its not mapped from any file, its zero here  
*00:00* is Major/Minor device number – Since its not mapped from any file, its zero here  
*0* is Inode number – Since its not mapped from any file, its zero here  
*[heap]*is Heap segment

**mmap**: malloc uses [mmap](http://lxr.free-electrons.com/source/mm/mmap.c?v=3.8#L1285) to create a private anonymous mapping segment. The primary purpose of private anonymous mapping is to allocate new memory (zero filled) and this new memory would be exclusively used by calling process.

*Example*:

/\* Private anonymous mapping example using mmap syscall \*/

#include <stdio.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

void static inline errExit(const char\* msg)

{

printf("%s failed. Exiting the process\n", msg);

exit(-1);

}

int main()

{

int ret = -1;

printf("Welcome to private anonymous mapping example::PID:%d\n", getpid());

printf("Before mmap\n");

getchar();

char\* addr = NULL;

addr = mmap(NULL, (size\_t)132\*1024, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);

if (addr == MAP\_FAILED)

errExit("mmap");

printf("After mmap\n");

getchar();

/\* Unmap mapped region. \*/

ret = munmap(addr, (size\_t)132\*1024);

if(ret == -1)

errExit("munmap");

printf("After munmap\n");

getchar();

return 0;

}

*Output Analysis*:

*Before mmap*: In the below output we can see only memory mapping segments that belongs to shared libraries libc.so and ld-linux.so

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6067/maps

08048000-08049000 r-xp 00000000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

08049000-0804a000 r--p 00000000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

0804a000-0804b000 rw-p 00001000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

**b7e21000-b7e22000 rw-p 00000000 00:00 0**

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$

*After mmap*: In the below output we can observe that our memory mapping segment (b7e00000 – b7e21000 whose size is 132KB) is combined with already existing memory mapping segment (b7e21000 – b7e22000).

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6067/maps

08048000-08049000 r-xp 00000000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

08049000-0804a000 r--p 00000000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

0804a000-0804b000 rw-p 00001000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

**b7e00000-b7e22000 rw-p 00000000 00:00 0**

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$

where  
*b7e00000-b7e22000* is Virtual address range for this segment  
*rw-p* is Flags (Read, Write, NoeXecute, Private)  
*00000000* is File offset – Since its not mapped from any file, its zero here  
*00:00* is Major/Minor device number – Since its not mapped from any file, its zero here  
*0* is Inode number – Since its not mapped from any file, its zero here

*After munmap*: In the below output we can see that our memory mapping segment is unmapped ie) its corresponding memory is released to the operating system.

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6067/maps

08048000-08049000 r-xp 00000000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

08049000-0804a000 r--p 00000000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

0804a000-0804b000 rw-p 00001000 08:01 539691 /home/sploitfun/ptmalloc.ppt/syscalls/mmap

**b7e21000-b7e22000 rw-p 00000000 00:00 0**

...

sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$

*NOTE*: In our sample program executions ASLR was turned off.

*Reference*:

1. [Anatomy of a program in memory](http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/)