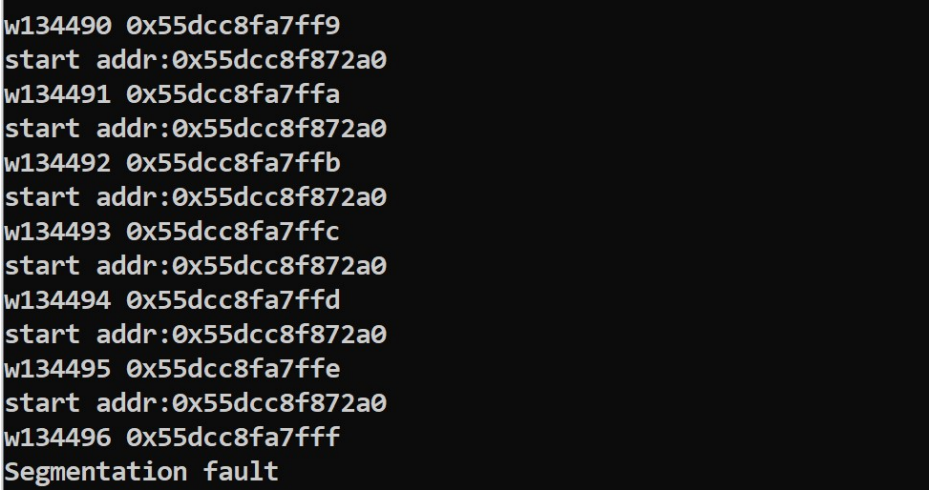


우선 예제 코드를 짜봤습니다

예제코드는 다음과 같은 구성으로 되어있습니다.

- 실제로 malloc(0); 할당 주소를 받는가?
- 할당 주소를 받는다면 얼마까지 수정이 가능한가?

예제	test1.c
	<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; #include &lt;time.h&gt; int main(void) {     int count = 0;     char* a = malloc(0);     time_t t;     printf("malloc(0) -&gt; %p\n", a);     for (char* s = a; ; s++) {         t = time(NULL);         *s = t % ('z' - 'a' + 1) + 'a';         count++;         printf("start addr:%p\n", a);         printf("%c%d %p\n", *s, count, s);     }     exit(0); }</pre>
결과	 <pre>w134490 0x55dcc8fa7ff9 start addr:0x55dcc8f872a0 w134491 0x55dcc8fa7ffa start addr:0x55dcc8f872a0 w134492 0x55dcc8fa7ffb start addr:0x55dcc8f872a0 w134493 0x55dcc8fa7ffc start addr:0x55dcc8f872a0 w134494 0x55dcc8fa7ffd start addr:0x55dcc8f872a0 w134495 0x55dcc8fa7ffe start addr:0x55dcc8f872a0 w134496 0x55dcc8fa7fff Segmentation fault</pre>

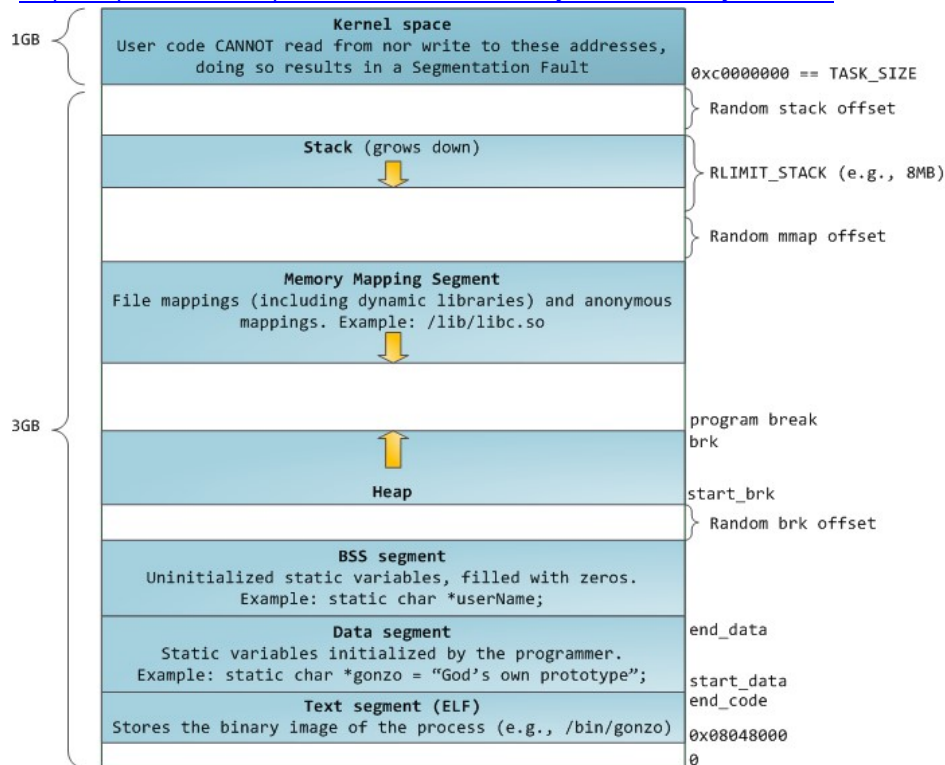
### <결과 분석>

- 해당 예제코드에서 malloc 을 하면 실제로 아무 크기도 할당받지 않은 char\* 포인터 주소 0x55dcc8f872a0 를 받는 점을 알 수 있습니다. 해당 주소는 Linux OS 기준으로 전형적으로 힙 주소의 가상공간의 주소로 리턴받고 있음을 확인했습니다.
- 또한 하나도 할당 받지 않은 char\* 포인터가 얼마만큼 데이터를 수정할 수 있는가 (page table 의 protect Bit 및 valid bit 검사) 를 해보았습니다. 해당 결과를 보니 시작점의 이유는 알 수 없으나 offset 이 0x2a0 로 시작하고, 끝점이 0xffff 임을 알 수 있습니다. 그 이후는 Segmentation Fault 가 발생합니다.

### <분석 예상>

1. 이에 대한 원인을 찾고자 몇 가지 공부를 해보니 malloc() 라이브러리 함수 내부에는 brk 와 mmap 시스템 콜을 호출하는데 이 때 **brk 시스템콜**을 살펴보니 내용을 알 수 있었습니다.
- malloc() 를 호출하면 가상주소공간의 heap 영역을 가리키는 포인터인 brk 를 증가 시키면서 할당하기 때문에 brk() 시스템 콜을 호출한다고 합니다.

[<https://sploitfun.wordpress.com/2015/02/11/syscalls-used-by-malloc/>]



- ※ 해당 리눅스 가상주소 정책에서 Random brk offset 을 통해 알 수 있는 점이 프로세스를 실행할 때 Random brk offset 을 랜덤하게 설정함으로써 heap 영역 시작 위치를 어렵게 만드는 **KASLR** (Kernel Address Layout Randomization) 정책을 사용함으로써 힙 주소공간이 프로세스 실행마다 바뀔 순 있지만 Heap 주소공간 영역은 어느정도 지키면서 배치됨 (0x55\*\*\*\*\*, \*는 Wildcard) 을 알 수 있습니다.

2. 그 중 리눅스 가상주소공간에서 heap 영역을 가리키는 포인터는 start\_brk 를 키워주는 시스템 콜이 brk() 이며 brk() 가 한 번 호출될 때마다 0x1000 만큼 영역이 증가함을 알 수 있습니다.

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

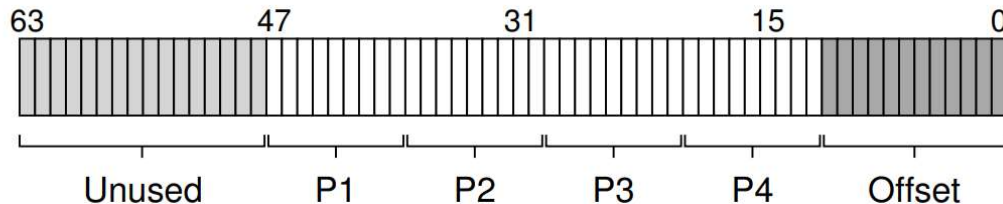
- start\_brk = end\_data = 0x804b000
- brk = 0x804c000.

•

```
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$
```

- [<https://sploitfun.wordpress.com/2015/02/11/syscalls-used-by-malloc/>]
- [<https://code1018.tistory.com/221>]

3. 왜 하필이면 아무것도 할당받지 않아도 하위 0xFFF 만큼 주소공간에 쓰기가 가능한지 생각을 해보았습니다. 제가 내린 일련의 결론은 리눅스 64 비트 사용중이지만 실제 OS 는 48 비트만 사용하므로 16 진수로 표기하면 12 자리수가 나옵니다. 그리고 4-level Page table 을 사용하고 있어 아래와 같은 그림으로 가상메모리 주소공간을 사용 중입니다.



[<https://pages.cs.wisc.edu/~remzi/OSTEP/vm-complete.pdf>]

- 중요한건 offset 이 12 비트라는 점인데 해당 비트가 16 진수로 뒤에서 3 칸을 의미하기 때문에 프로세스에서 첫 malloc 을 호출할 때, 페이지 4KB 크기만큼 할당해서 받아오기 때문에 아래와 같은 결론을 내릴 수 있을 것 같습니다.
- 리눅스의 모든 프로세스에선 첫 할당받은 heap 공간의 주소 끝의 0x000 ~ 0xFFF 까지는 할당가능하다.
- 예를들어, 첫 할당 받은 주소가 0x55898c6522a0 이라고 한다면 0x55898c652000 ~ 0x55898c652fff 까지는 임의로 작성하더라도 segmentation Fault 가 발생하지 않는다는 말을 의미합니다.
- 해당 실 예제를 직접 짜서 보면 아래와 같습니다..

```

여 -- test2.c -
제 // 힙으로 할당받아온 주소의 offset 을 자유롭게 조작하여 segment fault 를 띄우지 않고
   작동하게 하는 예제입니다

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define NON_OFFSET      0xFFFFFFFF000      //마지막 3 비트는 offset
#define START_OFF      0x001
#define END_OFF         0xFFF

#define HEAP_VAL(_VAL)  ((NON_OFFSET & (long int)_VAL) | START_OFF)

```

```
#define HEAP_VA_E(_VAL) ((NON_OFFSET & (long int)_VAL) | END_OFF)
```

```
int main(void)
{
    char *s = (char*)malloc(0);
    char *s1 = (char*)HEAP_VA_S(s);
    char *s2 = (char*)HEAP_VA_E(s);
    time_t t;

    printf("malloc memory addr : %p\n", s);
    printf("Look Prev Data\n");
    for (char* p = s1 ; p < s ; p++) {
        printf("%c", *p);
    }
    printf("\n");
    printf("start : %p, end : %p\n", s1, s2);

    for (char *ch = s1 ; s <= s2 ; s++) {
        t = time(NULL);
        *ch = t % ('z' - 'a' + 1) + 'a';
        printf("%c", *ch);
    }
    printf("\nExit!\n");
    exit(0);
}
```

결 malloc memory addr : 0x56474f8912a0

Look Prev Data

!

```
start : 0x56474f891001, end : 0x56474f891fff
```

[illegible]

	<pre> ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... ..... Exit! </pre>
	<p>마지막에 Exit! 가 찍힘으로보아 실제로 1 페이지 (4KB) 만큼은 힙 할당을 명시적으로 하지 않아도 사용할 수 있음을 볼 수 있습니다.</p>

예제 3	<p>&lt;실제로 malloc 으로 받지않은 공간에 대한 선처리를 할 수 있는가에 대한 예제입니다.&gt;</p> <pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt; #include &lt;string.h&gt;  #define NON_OFFSET      0xFFFFFFFF000      //마지막 3 비트는 offset #define START_OFF      0x001 #define END_OFF        0xFFF  #define START_ADD      0x440 #define SPACE_ADD      0x020 //메모리 증가추이가 0xN? 라면 --&gt; 0x(N+1)0 으로 증가한다. () /**  * 실제로 사용하려면 아래 경우의 수를 따져야한다.  * size 가 offset 크기 0xFFF 를 넘어가는 Case  * _count 가 20 보다 작다면 기본값으로 0x10 증가한 주소값이 아닌 0x20 증가한 값이 들어간다. </pre>
------	---

	<pre> * 초기할당하고 직후 malloc 을 시도하면 0x40 증가한다. */ #define NEXT_ADDR(_CUR, _COUNT)      ((( (long int)_CUR + (long int)_COUNT) &amp; 0xFFFFFFFFFFF0) + 0x10) #define HEAP_VAL_S(_VAL)              ((NON_OFFSET &amp; (long int)_VAL)   START_OFF) #define HEAP_VAL_E(_VAL)              ((NON_OFFSET &amp; (long int)_VAL)   END_OFF)  int main(void) {     int size = 0x42;     char* s1 = (char*)malloc(size);     char* s2 = (char*)malloc(size);     char* s3 = (char*)malloc(size);     char* s4 = (char*)malloc(size);     char *test = (char*)NEXT_ADDR(s4, size);     strcpy(test, "test set up");           // 할당 받기 전에 s5 조작 가능     char* s5 = (char*)malloc(size);       // 할당 받기 전에 조작 가능     printf("%p\n", s1);     printf("%p\n", s2);     printf("%p\n", s3);     printf("%p\n", s4);     printf("%p\n", s5);     printf("%s\n", s5);      exit(0); } </pre>
결과	<pre> 0x5588b4e9b2a0 0x5588b4e9b2f0 0x5588b4e9b340 0x5588b4e9b390 0x5588b4e9b3e0 test set up </pre>

제가 내린 결론은 이렇게 되는데 무언가 착각하고 있는 내용이 있는지 궁금합니다.

※또한 해당 내용을 분석하면서 추가적으로 궁금한 점이 생겼습니다.

- 위의 예제코드나 아래 예제코드나 동일하게 Heap 영역을 할당하였는데 힙 영역에 실제로 할당된건 malloc(0); 하나만 호출하였음에도 첫 brk 호출 시 offset 이 0x001 로 시작하지 않고 0x2a0 로 시작하는 이유가 궁금합니다.