viettel

Theo cách của bạn

Memory Layout of a Process

Trình bày: Bùi Duy Tiến

Contents

01

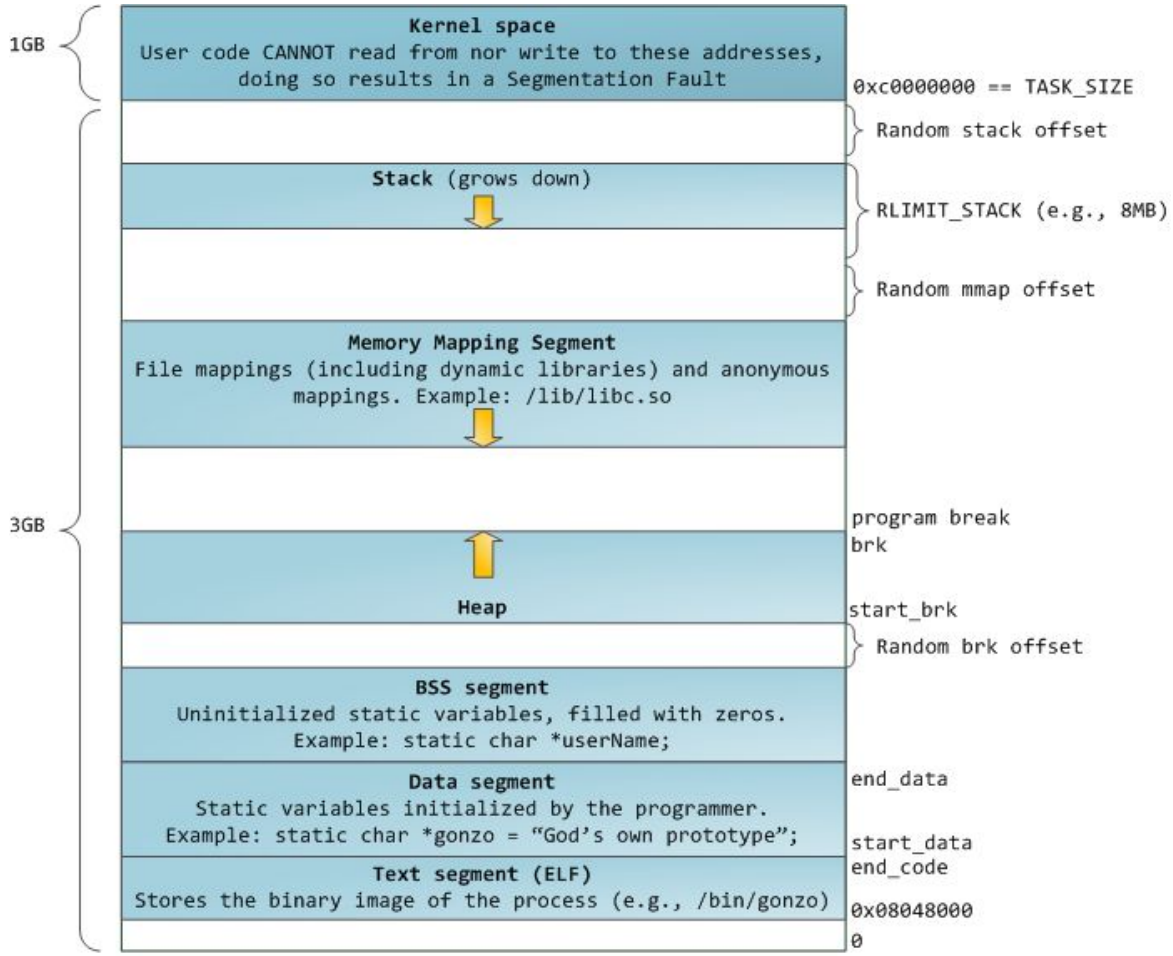
Memory layout of a Process

1. Memory layout of C

Khi tạo một chương trình và chạy chương trình, tệp thực thi của nó được lưu trữ một cách có tổ chức trong RAM (hoặc virtual memory) của máy tính.

Bố trí bộ nhớ cho chương trình C bao gồm các thành phần chính:

+ Text/Code Segment
+ Initialized Data Segment
+ Uninitialized Data Segment
+ Heap
+ Stack



1. Memory layout of C

Nội dung bộ nhớ của 1 chương trình

```c
#include <stdio.h>
#include <unistd.h>
int main(){
    printf("PID is: %d \n", getpid());
    while(1);
    return 0;
}
```

```
duytien@ubuntu:~/Workspace$ ./memory_size
PID is: 4186
```

```
duytien@ubuntu:~$ sudo cat /proc/4186/maps
[sudo] password for duytien:
00400000-00401000 r-xp 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00600000-00601000 r--p 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00601000-00602000 rw-p 00001000 08:01 264014                             /home/duytien/Workspace/memory_size
00b8a000-00bab000 rw-p 00000000 00:00 0                                  [heap]
7f23f3651000-7f23f3811000 r-xp 00000000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23f3811000-7f23f3a11000 ---p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23f3a11000-7f23f3a15000 r--p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23f3a15000-7f23f3a17000 rw-p 001c4000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23f3a17000-7f23f3a1b000 rw-p 00000000 00:00 0
7f23f3a1b000-7f23f3a41000 r-xp 00000000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f23f3c27000-7f23f3c2a000 rw-p 00000000 00:00 0
7f23f3c40000-7f23f3c41000 r--p 00025000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f23f3c41000-7f23f3c42000 rw-p 00026000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f23f3c42000-7f23f3c43000 rw-p 00000000 00:00 0
7ffd551da000-7ffd551fb000 rw-p 00000000 00:00 0                          [stack]
7ffd551fb000-7ffd551fe000 r--p 00000000 00:00 0                          [vvar]
7ffd551fe000-7ffd55200000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
duytien@ubuntu:~$ s
```

viettel

Theo cách của bạn

1. Memory layout of C

- Không có nội dung bộ nhớ trong khoảng: 0x00000000 - 0x00400000 = 4194304 byte = 4 Mb

- Địa chỉ bắt đầu 0x400000 được đặt bởi tập thực thi ELF

```c
#include <stdio.h>

int main () {
    void * addr = (void *) 0x0;
    printf("0x%x\n", ((char *) addr)[0]); // prints 0x0
    printf("0x%x\n", ((char *) addr)[1]); // prints 0x1
    printf("0x%x\n", ((char *) addr)[2]); // prints 0x2
    return 0;
}
```

```
duytien@ubuntu:~/Workspace$ gcc test.c -o test
duytien@ubuntu:~/Workspace$ ./test
Segmentation fault (core dumped)
duytien@ubuntu:~/Workspace$
```

/usr/lib/ldscripts/elf_x86_64.x

```
SECTIONS
{
  /* Read-only sections, merged into text segment: */
  PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x400000)); . = SE
GMENT_START("text-segment", 0x400000) + SIZEOF_HEADERS;
```

1. Memory Layout of C

+ 3 vùng nhớ Text segment, Initialize Data Segment, BSS đều được đặt kích thước mặc định là 4kb.
+ Đây là kích thước tối thiểu của 3 vùng này.

4096 bytes -Text
4096 bytes - IDS
4096 bytes - BSS

```
duytien@ubuntu:~$ sudo cat /proc/4186/maps
[sudo] password for duytien:
00400000-00401000 r-xp 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00600000-00601000 r--p 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00601000-00602000 rw-p 00001000 08:01 264014                             /home/duytien/Workspace/memory_size
```

02

Text/Code Segment

2. Text/Code Segment

+ Ở vùng nhớ có địa chỉ thấp nhất của chương trình, là nơi lưu trữ các mã lệnh đã được biên dịch của chương trình máy tính.

+ Chỉ chịu sự chi phối của hệ điều hành, các tác nhân khác không thể can thiệp trực tiếp đến phân vùng này.

+ Việc đưa các mã lệnh đã được biên dịch của chương trình lên code segment là công việc đầu tiên hệ điều hành làm khi chương trình chạy.

```c
#include <stdio.h>
#include <unistd.h>
int main(){
    printf("PID is: %d \n", getpid());
    while(1);
    return 0;
}
```

```
duytien@ubuntu:~/Workspace$ size memory_size
   text    data     bss     dec     hex filename
   1274     560       8    1842     732 memory_size
duytien@ubuntu:~/Workspace$
```

4096 bytes -Text
4096 bytes - IDS
4096 bytes - BSS

```
duytien@ubuntu:~$ sudo cat /proc/4186/maps
[sudo] password for duytien:
00400000-00401000 r-xp 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00600000-00601000 r--p 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00601000-00602000 rw-p 00001000 08:01 264014                             /home/duytien/Workspace/memory_size
```

viettel
Theo cách của bạn

2. Text/Code Segment

+ Tuy nhiên địa chỉ bộ nhớ bắt đầu (0x400000) không phải là địa chỉ bắt đầu của tệp thực thi.

+ Vị trí giữa 0x400000 và địa chỉ thực hiện tệp là cho ELF headers và Program Headers

```
duytien@ubuntu:~/Workspace$ readelf --file-header ./memory_size | grep 'Entry point address'
  Entry point address:               0x400470
duytien@ubuntu:~/Workspace$
```

```
duytien@ubuntu:~/Workspace$ objdump --disassemble-all --start-address=0x000000 --stop-address=0x401000 ./memory_size

./memory_size:     file format elf64-x86-64
```

```
Disassembly of section .text:

0000000000400470 <_start>:
  400470:       31 ed                   xor    %ebp,%ebp
  400472:       49 89 d1                mov    %rdx,%r9
  400475:       5e                      pop    %rsi
  400476:       48 89 e2                mov    %rsp,%rdx
  400479:       48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
  40047d:       50                      push   %rax
  40047e:       54                      push   %rsp
  40047f:       49 c7 c0 00 06 40 00    mov    $0x400600,%r8
  400486:       48 c7 c1 90 05 40 00    mov    $0x400590,%rcx
  40048d:       48 c7 c7 66 05 40 00    mov    $0x400566,%rdi
  400494:       e8 b7 ff ff ff          callq  400450 <__libc_start_main@plt>
  400499:       f4                      hlt
  40049a:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
```

03

Initialized Data Segment

3. Initialize Data Segment

+ Initialize data lưu trữ tất cả các biến glocal, static và external đã được khởi tạo giá trị.

```
duytien@ubuntu:~/Workspace$ size memory_size
   text    data     bss     dec     hex filename
   1274     568       8    1850     73a memory_size
duytien@ubuntu:~/Workspace$
```

```c
#include <stdio.h>
#include <unistd.h>
int a = 3;
int main(){
    static int b = 2;
    printf("Address of a %p \n", &a);
    printf("Address of b %p \n", &b);
    printf("PID is : %d \n", getpid());
    while(1);
    return 0;
}
```

```
Disassembly of section .data:

0000000000601038 <__data_start>:
        ...

0000000000601040 <__dso_handle>:
        ...

0000000000601048 <a>:
  601048:       03 00                   add    (%rax),%eax
        ...

000000000060104c <b.2782>:
  60104c:       02 00                   add    (%rax),%al
        ...

Disassembly of section .bss:

0000000000601050 <__bss_start>:
        ...
```

```
cs144@mininet-vm:~/Desktop$ ./size
Address of a 0x601048
Address of b 0x60104c
PID is : 3222
```

04

Uninitialized Data Segment

4. Uninitialized Data Segment (BSS)

Chứa các biến global, static được khởi tạo bằng 0 hoặc chưa được khởi tạo giá trị.

```
cs144@mininet-vm:~/Desktop$ ./size
Address of main function is: 0x40057d
Address of x 0x601058
Address of y 0x601054
Address of a 0x601048
Address of b 0x60104c
PID is : 3284
```

```
duytien@ubuntu:~/Workspace$ size memory_size
   text    data     bss     dec     hex filename
   1274     568      16    1858     742 memory_size
duytien@ubuntu:~/Workspace$
```

```c
#include <stdio.h>
#include <unistd.h>
int a = 3;
int x;

int main(){
    static int y = 0;
    static int b = 2;
    printf("Address of main function is: %p \n", &main);
    printf("Address of x %p \n", &x);
    printf("Address of y %p \n", &y);
    printf("Address of a %p \n", &a);
    printf("Address of b %p \n", &b);
    printf("PID is : %d \n", getpid());
    while(1);
    return 0;
}
```

```
Disassembly of section .data:

0000000000601038 <__data_start>:
        ...

0000000000601040 <__dso_handle>:
        ...

0000000000601048 <a>:
  601048:       03 00                   add    (%rax),%eax
        ...

000000000060104c <b.2784>:
  60104c:       02 00                   add    (%rax),%al
        ...

Disassembly of section .bss:

0000000000601050 <__bss_start>:
  601050:       00 00                   add    %al,(%rax)
        ...

0000000000601054 <y.2783>:
  601054:       00 00                   add    %al,(%rax)
        ...

0000000000601058 <x>:
        ...
cs144@mininet-vm:~/Desktop$
```

4. Uninitialized Data Segment (BSS)

Khi sử dụng quá vùng nhớ 4Mb, kernel sẽ tự động cấp phát thêm vùng nhớ có địa chỉ ở trên vùng BSS.

```c
#include <stdio.h>
#include <unistd.h>
int arr[10000000];
int main(){
    printf("Address of arr[0] %p \n", &(arr[0]));
    printf("Address of arr[10000000] %p \n", &(arr[10000000]));
    printf("PID is : %d \n", getpid());
    while(1);
    return 0;
}
```

Text
IDS
BSS

```
00400000-00401000 r-xp 00000000 08:01 419709                             /home/cs144/Desktop/size
00600000-00601000 r--p 00000000 08:01 419709                             /home/cs144/Desktop/size
00601000-00602000 rw-p 00001000 08:01 419709                             /home/cs144/Desktop/size
00602000-02c27000 rw-p 00000000 00:00 0
```

```
cs144@mininet-vm:~/Desktop$ ./size
Address of arr[0] 0x601080
Address of arr[10000000] 0x2c26a80
PID is : 2863
```

05

Stack

5. Stack

+ Stack là một vùng nhớ được cấp phát tự động và có cấu trúc LIFO.

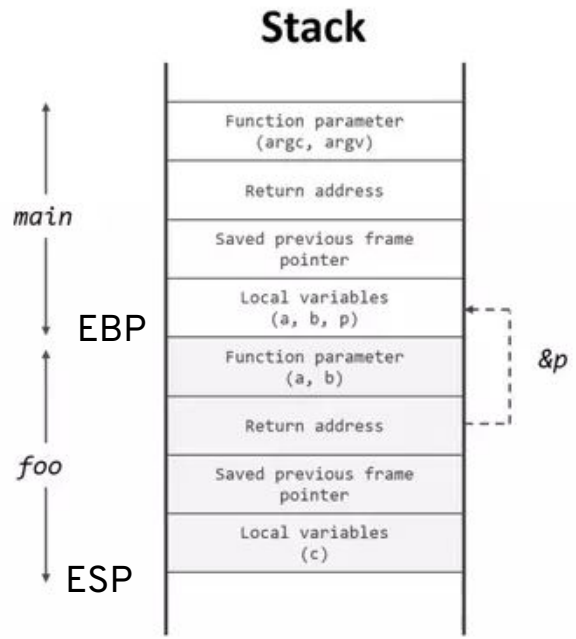
+ Khi một hàm được gọi, stack frame tương ứng của hàm đó sẽ được push vào stack.

+ Một stack frame có cấu trúc gồm:
  - Function parameter
  - Return address
  - Saved previous frame pointer
  - Local variable

+ Các stack frame tương ứng với một hàm sẽ bị xóa và bộ nhớ chúng sử dụng được giải phóng sau khi hàm chạy xong.

```
int foo(int a, int b) {
  int c = 10;
  return c + a * b;
}

int main(int argc, char *argv[]) {
  int a = 5, b = 6, p;
  p = foo(a, b);
  return 0;
}
```



5.1 Stack overflow

+ Thông thường, kích thước stack bị giới hạn.
  - Linux 64 bit: 8Mb
  - Window 64 bit: 1Mb

+ Nếu một chương trình sử dụng nhiều không gian bộ nhớ hơn kích thước ngăn xếp thì hiện tượng stack overflow sẽ xảy ra.

```
cs144@mininet-vm:~$ ulimit -s
8192
```

```
ontMac:~ TTTD$ ulimit -s
8192
```

5.1 Stack Overflow (cont.)

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

int main(){
    int a[2000000];
    printf("PID is %d \n", getpid());
    while(1);
    return 0;
}
```

```
duytien@ubuntu:~/Workspace$ gcc stack_size.c -o stack_size
duytien@ubuntu:~/Workspace$ ./stack_size
PID is 4901
```

```
duytien@ubuntu:~$ sudo cat /proc/4901/maps
[sudo] password for duytien:
00400000-00401000 r-xp 00000000 08:01 262126                             /home/duytien/Workspace/stack_size
00600000-00601000 r--p 00000000 08:01 262126                             /home/duytien/Workspace/stack_size
00601000-00602000 rw-p 00001000 08:01 262126                             /home/duytien/Workspace/stack_size
020e5000-02106000 rw-p 00000000 00:00 0                                  [heap]
7f86a31c7000-7f86a3387000 r-xp 00000000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f86a3387000-7f86a3587000 ---p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f86a3587000-7f86a358b000 r--p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f86a358b000-7f86a358d000 rw-p 001c4000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f86a358d000-7f86a3591000 rw-p 00000000 00:00 0
7f86a3591000-7f86a35b7000 r-xp 00000000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f86a379d000-7f86a37a0000 rw-p 00000000 00:00 0
7f86a37b6000-7f86a37b7000 r--p 00025000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f86a37b7000-7f86a37b8000 rw-p 00026000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f86a37b8000-7f86a37b9000 rw-p 00000000 00:00 0
7fff0d6b6000-7fff0de5a000 rw-p 00000000 00:00 0                          [stack]
7fff0df92000-7fff0df95000 r--p 00000000 00:00 0                          [vvar]
7fff0df95000-7fff0df97000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
duytien@ubuntu:~$
```

5.1 Stack Overflow (cont.)

```c
#include <stdio.h>

int main(){
    printf("Hello /n");
    int a[2500000];
    return 0;
}
```

Có hai trường hợp có thể xảy ra tràn ngăn xếp:

+ Khai báo số lượng lớn các biến cục bộ, một mảng hoặc ma trận có kích thước lớn dẫn đến tràn ngăn xếp.

+ Hàm đệ quy vô hạn.

```c
#include <stdio.h>

void func (int x){
    if (x == 1){
        return;
    }
    x = 6;
    func (x);
}
int main (){
    int x = 5;
    func (x);
}

```

Kết quả: Trả về lỗi Segmentation Fault

```
duytien@ubuntu:~/Workspace$ gcc stack_size.c -o stack_size
duytien@ubuntu:~/Workspace$ ./stack_size
Segmentation fault (core dumped)
duytien@ubuntu:~/Workspace$
```

5.1 Stack Overflow (cont.)

- Tuy nhiên các giá trị này chỉ là mặc định, có thể yêu cầu phân bổ thêm để thay đổi giới hạn không gian stack.

- Có một số cách để thay đổi giới hạn stack:
  + ulimit –s
  + Dùng getrlimit() và setrlimit system calls.

- Việc phân bố thêm bộ nhớ cho stack vẫn chịu giới hạn là RAM (hoặc virtual memory)

```
duytien@ubuntu:~$ ulimit -s
8192
duytien@ubuntu:~$ ulimit -s 16384
duytien@ubuntu:~$ ulimit -s
16384
duytien@ubuntu:~$
```

```
duytien@ubuntu:~/Workspace$ gcc stack_size.c -o stack_size
duytien@ubuntu:~/Workspace$ ./stack_size
PID is 5334
```

```
7ffdf2c53000-7ffdf37c8000 rw-p 00000000 00:00 0                          [stack]
```

5.1 Stack Overflow (cont.)

```
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

```
struct rlimit {
    rlim_t rlim_cur;  /* Soft limit */
    rlim_t rlim_max;  /* Hard limit (ceiling for rlim_cur) */
};
```

```c
#include <sys/resource.h>
#include <stdio.h>
#include <unistd.h>

int main (int argc, char **argv)
{
    const rlim_t kStackSize = 64 * 1024 * 1024;   // min stack size = 64 Mb
    struct rlimit rl;
    int result;

    result = getrlimit(RLIMIT_STACK, &rl);
    if (result == 0)
    {
        if (rl.rlim_cur < kStackSize)
        {
            rl.rlim_cur = kStackSize;
            result = setrlimit(RLIMIT_STACK, &rl);
            if (result != 0)
            {
                fprintf(stderr, "setrlimit returned result = %d\n", result);
            }
        }
    }
    int a[3000000];
    printf("PID is %d \n", getpid());
    while(1);
    return 0;
}
```

06

Heap

6. Heap

- Heap là đoạn diễn ra việc cấp phát bộ nhớ động.
- Heap bắt đầu từ cuối đoạn BSS và mở rộng lên các địa chỉ lớn hơn.
- Vùng heap được quản lý bằng malloc(), realloc() và free.
- Khi yêu cầu nhiều bộ nhớ hơn kích thước có sẵn ở heap, nó sẽ yêu cầu kernel cung cấp thêm bằng cách sử dụng system call brk(), sbrk() hoặc mmap().

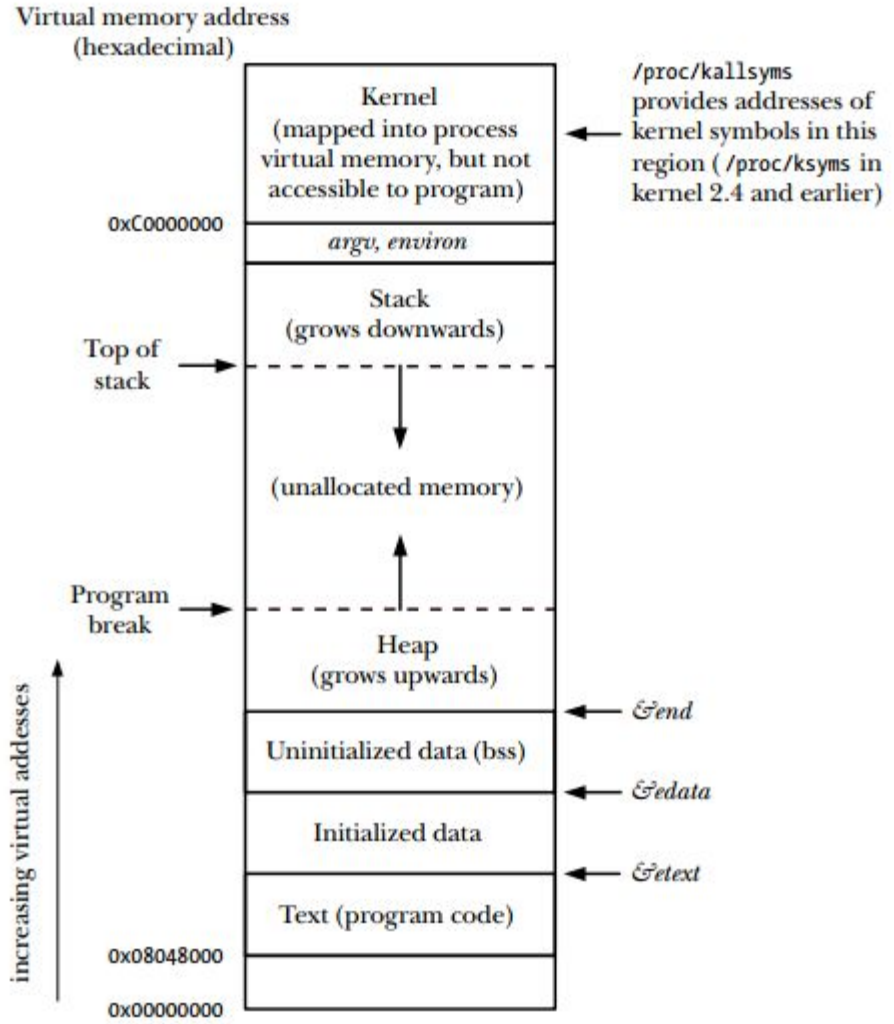
```
duytien@ubuntu:~$ sudo cat /proc/4585/maps
00400000-00401000 r-xp 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00600000-00601000 r--p 00000000 08:01 264014                             /home/duytien/Workspace/memory_size
00601000-00602000 rw-p 00001000 08:01 264014                             /home/duytien/Workspace/memory_size
01584000-015a5000 rw-p 00000000 00:00 0                                  [heap]
```

6. Heap

- System call brk(), sbrk()

```
#include <unistd.h>

int brk(void *end_data_segment);
                                        Returns 0 on success, or -1 on error
void *sbrk(intptr_t increment);
                Returns previous program break on success, or (void *) -1 on error
```



6.1 Heap Overflow

- Vùng nhớ Heap không có giới hạn mặc định mà chỉ có giới hạn duy nhất là RAM (hoặc virtual memory).
- Có hai trường hợp có thể dẫn đến heap overflow :

+ Liên tục cấp phát bộ nhớ nhưng không giải phóng sau khi sử dụng.

+ Cấp phát động một số lượng lớn các biến.

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    for(int i = 0; i < 100000000; i++){
        int *ptr = (int*) calloc(10, sizeof(int));
    }
    return 0;
}
```

```
duytien@ubuntu:~/Workspace$ gcc heap_overflow.c -o heap_overflow
duytien@ubuntu:~/Workspace$ ./heap_overflow
Killed
duytien@ubuntu:~/Workspace$
```

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *ptr = (int*)calloc(100000000000, sizeof(int));
    return 0;
}
```

```
cs144@mininet-vm:~$ ./heap
(nil)
cs144@mininet-vm:~$
```

6.1 Heap Overflow

```
duytien@ubuntu:~$ cat /proc/6589/maps
00400000-00401000 r-xp 00000000 08:01 285363                             /home/duytien/Workspace/memory_size
00600000-00601000 r--p 00000000 08:01 285363                             /home/duytien/Workspace/memory_size
00601000-00602000 rw-p 00001000 08:01 285363                             /home/duytien/Workspace/memory_size
00f50000-00f71000 rw-p 00000000 00:00 0                                  [heap]
7f2f033bd000-7f2f0357d000 r-xp 00000000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f2f0357d000-7f2f0377d000 ---p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f2f0377d000-7f2f03781000 r--p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f2f03781000-7f2f03783000 rw-p 001c4000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f2f03783000-7f2f03787000 rw-p 00000000 00:00 0
```

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

int main(){
    printf("PID is: %d \n", getpid());
    while(1);
    return 0;
}
```

- Vùng nhớ có định danh heap vẫn giữ nguyên kích thước.
- Một vùng nhớ ẩn danh được tạo ngay trên heap có kích thước 3.82 Mb

```
duytien@ubuntu:~$ cat /proc/6702/maps
00400000-00401000 r-xp 00000000 08:01 285357                             /home/duytien/Workspace/memory_size
00600000-00601000 r--p 00000000 08:01 285357                             /home/duytien/Workspace/memory_size
00601000-00602000 rw-p 00001000 08:01 285357                             /home/duytien/Workspace/memory_size
022d3000-022f4000 rw-p 00000000 00:00 0                                  [heap]
7f57ed1a2000-7f57ed573000 rw-p 00000000 00:00 0
7f57ed573000-7f57ed733000 r-xp 00000000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f57ed733000-7f57ed933000 ---p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f57ed933000-7f57ed937000 r--p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f57ed937000-7f57ed939000 rw-p 001c4000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
```

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

int main(){
    int *ptr = (int*)malloc(1000000*sizeof(int));
    printf("PID is: %d \n", getpid());
    while(1);
    return 0;
}
```

6.1 Heap Overflow

```
duytien@ubuntu:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3921         411        3252          13         257        3227
Swap:           974         462         512
duytien@ubuntu:~$
```

```c
int main(int argc,char* argv[]){
        size_t oneHundredMiB=100*1048576;
        size_t maxMemMiB=0;
        void *memPointer = NULL;
        do{
                if(memPointer != NULL){
                        printf("Max Tested Memory = %zi\n",maxMemMiB);
                        memset(memPointer,0,maxMemMiB);
                        free(memPointer);
                }
                maxMemMiB+=oneHundredMiB;
                memPointer=malloc(maxMemMiB);
        }while(memPointer != NULL);
        printf("Max Usable Memory aprox = %zi\n",maxMemMiB-oneHundredMiB);
        return 0;
}
```

```
Max Usable Memory aprox = 3984588800
```

7. Memory Mapping Segment

- Chứa các tệp Shared lib có đuôi .so
- Có thể tạo ánh xạ bộ nhớ ẩn danh không tương ứng với bất kỳ tệp nào, được sử dụng để thay thế dữ liệu trong chương trình.
- Khi chạy nhiều lần kiểm tra địa chỉ của process, các địa chỉ của thư viện thay đổi do trình liên kết thực hiện ngẫu nhiên hóa địa chỉ.

```
7f23ec42d000-7f23ec5ed000 r-xp 00000000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23ec5ed000-7f23ec7ed000 ---p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23ec7ed000-7f23ec7f1000 r--p 001c0000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23ec7f1000-7f23ec7f3000 rw-p 001c4000 08:01 428560                     /lib/x86_64-linux-gnu/libc-2.23.so
7f23ec7f3000-7f23ec7f7000 rw-p 00000000 00:00 0
7f23ec7f7000-7f23ec81d000 r-xp 00000000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f23eca03000-7f23eca06000 rw-p 00000000 00:00 0
7f23eca1c000-7f23eca1d000 r--p 00025000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
7f23eca1d000-7f23eca1e000 rw-p 00026000 08:01 428552                     /lib/x86_64-linux-gnu/ld-2.23.so
```