Illustrating /proc/xxx/maps and [vdso] in Linux

Three methods are used to analyze the address space of the process:

First, the size utility is used to list the section sizes of the object file:

```
$ size -A -x b.out
section         size        addr
.text           0x488       0x400a30
.data           0x420       0x602080
.bss            0x648       0x6024a0
```

Secondly, **etext**, **edata** and **end** variables, each of which points to the first address passed the end of the text segment, data segment and bss segment respectively, are checked. **sbrk(0)** is also used to get the end of the heap segment.

```
End of text     -> 000000400ec6
End of data     -> 0000006024a0
End of bss      -> 000000602ae8
End of heap     -> 000000f5e000
```

We can use the data in the output of the size utility to calculate the end of the three sections:

```
End of text     = 0x400a30+0x488 = 0x400eb8
End of data     = 0x602080+0x420 = 0x6024a0
End of bss      = 0x6024a0+0x648 = 0x602ae8
```

The results are almost identical to **etext**, **edata**, **end** and **sbrk(0)**.

Finally, the /proc/self/maps is read, and the whole memory map is printed out.

We see that while the size of the text segment is only 1128 bytes, the whole page is reserved for the text segment (the page size is 4096 bytes):

```
00400000-00402000 r-xp 00000000 00:14 21257691                  /tmp/b.out
```

The “r” in the permission bits means that the region can be read, and the “x” stands for execution permission, which is required since this section contains the program’s instructions.

The addresses of the functions we checked are all within this section, and notice that the constant array is also in this section:

```
&main = 000000400b24,       &func = 000000400b14,
&printf = 000000400928,     &scanf = 0000004009d8,
&ct[0] = 000000401180
```

The next section is the data segment.

```
00601000-00602000 r--p 00001000 00:14 21257691                  /tmp/b.out
&it[0] = 0000006020a0
```

And the bss segment:
00602000-00603000 rw-p 00002000 00:14 21257691 /tmp/b.out
&dt[0] = 0000006026e0, &cin = 0000006024a0, &cout = 0000006025c0

The heap segment:
00f3d000-00f5e000 rw-p 00000000 00:00 0 [heap]
&hp[0] = 000000f3d010

The stack segment:
7fff31a3e000-7fff31a61000 rw-p 00000000 00:00 0 [stack]
&st[0] = 7fff31a5e710

Virtual Dynamically-linked Shared Object

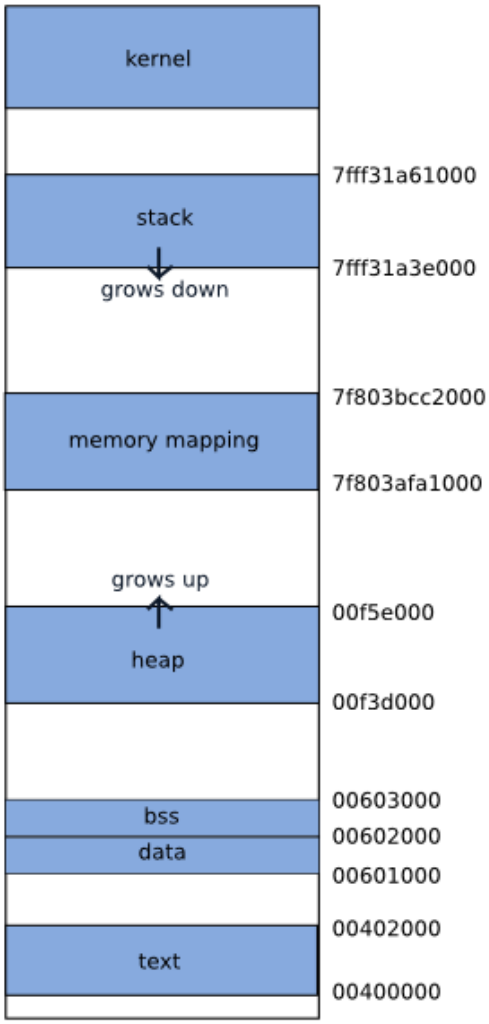
VDSO is a kernel-provided shared library that exports kernel space routines to user space applications, using standard mechanisms for linking and loading.
It helps to reduce the calling overhead on system calls, and also can work as a way to select the most efficient syscall mechanism.

Program Output

---------Virtual addresses of variables---------
&ct[0] = 000000401180
&it[0] = 0000006020a0
&dt[0] = 0000006026e0
&st[0] = 7fff31a5e710
&hp[0] = 000000f3d010

---------Virtual addresses of functions---------
&main = 000000400b24
&func = 000000400b14
&printf = 000000400928
&scanf = 0000004009d8

---------Virtual addresses of C++ iostreams---------
&cin = 0000006024a0
&cout = 0000006025c0



---------Addresses of program segments---------

First address past:

program text -> 000000400ec6

initialized data -> 0000006024a0

uninitialized data -> 000000602ae8

heap -> 000000f5e000

---------Memory map---------

00400000-00402000 r-xp 00000000 00:14 21257691 /tmp/b.out

00601000-00602000 r--p 00001000 00:14 21257691 /tmp/b.out

00602000-00603000 rw-p 00002000 00:14 21257691 /tmp/b.out

00f3d000-00f5e000 rw-p 00000000 00:00 0 [heap]

7f803afa1000-7f803b0f1000 r-xp 00000000 09:03 72084556 /lib64/libc-2.11.2.so

7f803b0f1000-7f803b2f0000 ---p 00150000 09:03 72084556 /lib64/libc-2.11.2.so

7f803b2f0000-7f803b2f4000 r--p 0014f000 09:03 72084556 /lib64/libc-2.11.2.so

7f803b2f4000-7f803b2f5000 rw-p 00153000 09:03 72084556 /lib64/libc-2.11.2.so

7f803b2f5000-7f803b2fa000 rw-p 00000000 00:00 0

7f803b2fa000-7f803b310000 r-xp 00000000 09:03 100918143 /lib64/libgcc_s.so.1

7f803b310000-7f803b50f000 ---p 00016000 09:03 100918143 /lib64/libgcc_s.so.1

7f803b50f000-7f803b510000 r--p 00015000 09:03 100918143 /lib64/libgcc_s.so.1

7f803b510000-7f803b511000 rw-p 00016000 09:03 100918143 /lib64/libgcc_s.so.1

7f803b511000-7f803b591000 r-xp 00000000 09:03 72084558 /lib64/libm-2.11.2.so

7f803b591000-7f803b790000 ---p 00080000 09:03 72084558 /lib64/libm-2.11.2.so

7f803b790000-7f803b791000 r--p 0007f000 09:03 72084558 /lib64/libm-2.11.2.so

7f803b791000-7f803b792000 rw-p 00080000 09:03 72084558 /lib64/libm-2.11.2.so

7f803b792000-7f803b884000 r-xp 00000000 09:03 916054 /usr/lib64/gcc/x86_64-pc-linux-gnu/4.4.3/libstdc++.so.6.0.13

7f803b884000-7f803ba84000 ---p 000f2000 09:03 916054 /usr/lib64/gcc/x86_64-pc-linux-gnu/4.4.3/libstdc++.so.6.0.13

7f803ba84000-7f803ba8b000 r--p 000f2000 09:03 916054 /usr/lib64/gcc/x86_64-pc-linux-gnu/4.4.3/libstdc++.so.6.0.13

7f803ba8b000-7f803ba8d000 rw-p 000f9000 09:03 916054 /usr/lib64/gcc/x86_64-pc-linux-gnu/4.4.3/libstdc++.so.6.0.13

7f803ba8d000-7f803baa2000 rw-p 00000000 00:00 0

7f803baa2000-7f803bac0000 r-xp 00000000 09:03 72084555 /lib64/ld-2.11.2.so

7f803bca5000-7f803bca9000 rw-p 00000000 00:00 0

7f803bcad000-7f803bcbf000 rw-p 00000000 00:00 0

7f803bcbf000-7f803bcc0000 r--p 0001d000 09:03 72084555 /lib64/ld-2.11.2.so

7f803bcc0000-7f803bcc1000 rw-p 0001e000 09:03 72084555 /lib64/ld-2.11.2.so

7f803bcc1000-7f803bcc2000 rw-p 00000000 00:00 0

7fff31a3e000-7fff31a61000 rw-p 00000000 00:00 0 [stack]

7fff31bff000-7fff31c00000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Source Code

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <climits>
using namespace std;
const size_t SZ = 1024;
char dt[SZ];
const char ct[SZ] = "an const string.";
char it[SZ] = "an initialized string.";
void func() { printf("This is a function.\n");}
typedef unsigned long long int addr_t;
int main()
{
        char st[SZ];
        char *hp = static_cast<char *>(malloc(SZ*sizeof(char)));
        FILE *pmap = fopen("/proc/self/maps", "r");
        extern char etext, edata, end;
        void *eheap = sbrk(0);
        memset(dt, 0xff, sizeof(dt));
        memset(st, 0xff, sizeof(st));
        memset(hp, 0xff, SZ*sizeof(char));
        printf("---------Virtual addresses of variables---------\n");
        printf("&ct[0] = %012llx\n", (addr_t)&ct[0]);
        printf("&it[0] = %012llx\n", (addr_t)&it[0]);
        printf("&dt[0] = %012llx\n", (addr_t)&dt[0]);
        printf("&st[0] = %012llx\n", (addr_t)&st[0]);
        printf("&hp[0] = %012llx\n", (addr_t)&hp[0]);
        printf("\n\n---------Virtual addresses of functions---------\n");
        printf("&main = %012llx\n", (addr_t)&main);
        printf("&func = %012llx\n", (addr_t)&func);
        printf("&printf = %012llx\n", (addr_t)&printf);
        printf("&scanf = %012llx\n", (addr_t)&scanf);
        printf("\n\n---------Virtual addresses of C++ iostreams---------\n");
        printf("&cin = %012llx\n", (addr_t)&cin);
        printf("&cout = %012llx\n", (addr_t)&cout);
        printf("\n\n---------Addresses of program segments---------\n");
```

```
        printf("First address past:\n");
        printf("program text -> %012llx\n", (addr_t)&etext);
        printf("initialized data -> %012llx\n", (addr_t)&edata);
        printf("uninitialized data -> %012llx\n", (addr_t)&end);
        printf("heap -> %012llx\n", (addr_t)eheap);
        printf("\n\n---------Memory map---------\n");
        char c;
        while ((c = getc(pmap)) != EOF)
                putchar(c);
        fclose(pmap);
        free(hp);
        return 0;
}
```