第十六章-分段

预备知识

`segmentation.py`使您可以查看在带分段的系统中如何执行地址转换。
该系统使用的分段非常简单：地址空间只有两个段；
此外，由进程生成的虚拟地址的高位确定该地址位于哪个段中：
0 为段 0（例如，代码和堆将驻留在其中），
1 为段 1（栈位于其中）。
段 0 沿正方向（朝更高的地址）增长，而段 1 沿负方向增长。

在视觉上，地址空间如下所示：

```
 --------------- virtual address 0
 |    seg0      |
 |              |
 |              |
 |--------------|
 |              |
 |              |
 |              |
 |              |
 |(unallocated)|
 |              |
 |              |
 |              |
 |--------------|
 |              |
 |    seg1      |
 |--------------| virtual address max (size of address space)
```

您可能还记得分段的情况，每个分段都有一对基址/界限寄存器。
因此，在这个问题中，存在两对基址/界限寄存器。
段 0 的基址说明段 0 的顶部地址(top)在物理内存中的位置,而界限寄存器则表明段的大小。
段 1 的基址说明段 1 的底部(bottom)在物理内存中的位置，而界限寄存器也表明段的大小（或它在负方向上增长了多少）。

和以前一样，有两个步骤可以运行该程序以测试您对分段的理解。 首先，在不带" -c"标志的情况下运行以生成一组地址，并查看您是否可以自己正确执行地址转换。
然后，完成后，使用" -c"参数运行以检查答案。

运行

```
python2 segmentation.py -h
```

获取所有参数

```
python2 segmentation.py -h
Usage: segmentation.py [options]

Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -A ADDRESSES, --addresses=ADDRESSES
                        a set of comma-separated pages to access; -1 means
                        randomly generate
  -a ASIZE, --asize=ASIZE
                        address space size (e.g., 16, 64k, 32m, 1g)
  -p PSIZE, --physmem=PSIZE
                        physical memory size (e.g., 16, 64k, 32m, 1g)
  -n NUM, --numaddrs=NUM
                        number of virtual addresses to generate
  -b BASE0, --b0=BASE0  value of segment 0 base register
  -l LEN0, --l0=LEN0    value of segment 0 limit register
  -B BASE1, --b1=BASE1  value of segment 1 base register
  -L LEN1, --l1=LEN1    value of segment 1 limit register
  -c                    compute answers for me
```

Problem1

问题描述

先让我们用一个小地址空间来转换一些地址。这里有一组简单的参数和几个不同的随机种子。你可以转换这些地址吗?

```
segmentation.py -a 128 -p 512 -b 0 -1 20 -B 512 -L 20 -s 0
segmentation.py -a 128 -p 512 -b 0 -1 20 -B 512 -L 20 -s 1
segmentation.py -a 128 -p 512 -b 0 -1 20 -B 512 -L 20 -s 2
```

问题分析

```
根据**VA**的**topbit**确定**segment**
地址空间是128字节，需要7位来表示，所以**topbit**是第**7**位(从第1位起)

对于**segment 0**沿正方向增长
offset=VA
0≤offset≤limit-1
对于**segment 1**沿负方向增长
offset=VA%max_size-maxsize
-limit≤offset≤-1

PA=base+offset
```

问题解答1

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> PA or segmentation violation?
  VA  1: 0x00000061 (decimal:   97) --> PA or segmentation violation?
  VA  2: 0x00000035 (decimal:   53) --> PA or segmentation violation?
  VA  3: 0x00000021 (decimal:   33) --> PA or segmentation violation?
  VA  4: 0x00000041 (decimal:   65) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment
0
grows in the positive direction, whereas segment 1 in the negative.
```

- VA 0: 0x0000006c (1101100B) topbit=1 segment 1
  offset=VA%max_size-maxsize=108%64-64=-20
  PA=base+offset=512-20=492

- VA 1: 0x00000061 (1100001B) topbit=1 segment 1
  offset=VA%max_size-maxsize=97%64-64=-31<-20
  segmentation violation

- VA 2: 0x00000035 (0110101B) topbit=0 segment 0
  offset=VA=53>20
  segmentation violation

- VA 3: 0x00000021 (0100001B) topbit=0 segment 0
  offset=VA=33>20
  segmentation violation

- VA 4: 0x00000041 (1000001B) topbit=1 segment 1
  offset=VA%max_size-maxsize=65%64-64=-63<-20

segmentation violation

答案验证1

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001e (decimal:  492)
  VA  1: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x00000035 (decimal:   53) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000021 (decimal:   33) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000041 (decimal:   65) --> SEGMENTATION VIOLATION (SEG1)
```

经验证，结果正确。

问题解答2

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> PA or segmentation violation?
  VA  1: 0x0000006c (decimal:  108) --> PA or segmentation violation?
  VA  2: 0x00000061 (decimal:   97) --> PA or segmentation violation?
  VA  3: 0x00000020 (decimal:   32) --> PA or segmentation violation?
  VA  4: 0x0000003f (decimal:   63) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
```

```
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment
0
grows in the positive direction, whereas segment 1 in the negative.
```

- VA 0: 0x00000011 (0010001B) topbit=0 segment 0
  offset=VA=17<20
  PA=base+offset=0+17=17
- VA 1: 0x0000006c (1101100B) topbit=1 segment 1
  offset=VA%max_size-maxsize=108%64-64=-20
  PA=base+offset=512-20=492
- VA 2: 0x00000061 (1100001B) topbit=1 segment 1
  offset=VA%max_size-max_size=97%64-64=-31<-20
  segmentation violation
- VA 3: 0x00000020 (0100000B) topbit=0 segment 0
  offset=VA=32>20
  segmentation violation
- VA 4: 0x0000003f (0111111B) topbit=0 segment 0
  offset=VA=63>20
  segmentation violation

答案验证2

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1 -c
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> VALID in SEG0: 0x00000011 (decimal:   17)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  2: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x00000020 (decimal:   32) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x0000003f (decimal:   63) --> SEGMENTATION VIOLATION (SEG0)
```

经验证，结果正确。

问题解答3

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2
ARG seed 2
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> PA or segmentation violation?
  VA  1: 0x00000079 (decimal:  121) --> PA or segmentation violation?
  VA  2: 0x00000007 (decimal:    7) --> PA or segmentation violation?
  VA  3: 0x0000000a (decimal:   10) --> PA or segmentation violation?
  VA  4: 0x0000006a (decimal:  106) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment
0
grows in the positive direction, whereas segment 1 in the negative.
```

- VA 0: 0x0000007a (1111010B) topbit=1 segment 1
  offset=VA%max_size-maxsize=122%64-64=-6>-20
  PA=base+offset=512-6=506

- VA 1: 0x00000079 (1111001B) topbit=1 segment 1
  offset=VA%max_size-maxsize=121%64-64=-7>-20
  PA=base+offset=512-7=505

- VA 2: 0x00000007 (0000111B) topbit=0 segment 0
  offset=VA=7<20
  PA=base+offset=0+7=7

- VA 3: 0x0000000a (0001010B) topbit=0 segment 0
  offset=VA=10<20
  PA=base+offset=0+10=10

- VA 4: 0x0000006a (1101010B) topbit=1 segment 1
  offset=VA%max_size-maxsize=106%64-64=-22<-20
  segmentation violation

答案验证3

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2 -c
ARG seed 2
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> VALID in SEG1: 0x000001fa (decimal:  506)
  VA  1: 0x00000079 (decimal:  121) --> VALID in SEG1: 0x000001f9 (decimal:  505)
  VA  2: 0x00000007 (decimal:    7) --> VALID in SEG0: 0x00000007 (decimal:    7)
  VA  3: 0x0000000a (decimal:   10) --> VALID in SEG0: 0x0000000a (decimal:   10)
  VA  4: 0x0000006a (decimal:  106) --> SEGMENTATION VIOLATION (SEG1)
```

经验证，结果正确。

Problem2

问题描述

现在，让我们看看是否理解了这个构建的小地址空间(使用上面问题的参数)
段 0 中最高的合法虚拟地址是什么?
段 1 中最低的合法虚拟地址是什么?
在整个地址空间中,最低和最高的非法地址是什么?
最后,如何运行带有 A 标志的 segmentation.py 来测试你是否正确?

问题分析

```
  --------------- virtual address 0
  |    seg0      |
  |              |
  |              |
  |-------------|<----段 0 1 中最高的合法虚拟地址 | |(unallocated)|<----非法地址
 |-------------|<----段 中最低的合法虚拟地址 seg1 |-------------| virtual
address max (size of space) < pre>
```

问题解答

段 0 中最高的合法虚拟地址=0+l-1=19,

段 1 中最低的合法虚拟地址=virtual address max-L=108

在整个地址空间中，最低和最高的非法地址是 20,107

答案验证

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -A
19,108,20,107 -c
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000013 (decimal:   19) --> VALID in SEG0: 0x00000013 (decimal:   19)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  2: 0x00000014 (decimal:   20) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x0000006b (decimal:  107) --> SEGMENTATION VIOLATION (SEG1)
```

经验证，结果正确。

Problem3

问题描述

假设我们在一个 128 字节的物理内存中有一个很小的 16 字节地址空间。你会设置什么样的基址和界限,以便让模拟器为指定的地址流生成以下转换结果:有效,有效,违规,违规,有效,有效?假设用以下参数:

```
segmentation.py -a 16 -p 128
  -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
  --b0 ? --l0 ? --b1 ? --l1 ?
```

注: 原书问题为:valid, valid, violation, ..., violation, valid, valid,即要求 0,1,14,15 有效,其余无效。

问题分析

```
 --------------- virtual address 0
 |    seg0     |
 |             |
 |             |
 |-------------|<----段 0 1 中最高的合法虚拟地址 | |(unallocated)|<----非法地址
|-------------|<----段 中最低的合法虚拟地址 seg1 |-------------| virtual
address max (size of space) < pre>
```

问题解答

不难得出
段 0 中最高的合法虚拟地址=1
段 1 中最低的合法虚拟地址=14

又因为
段 0 中最高的合法虚拟地址=0+l0-1
段 1 中最低的合法虚拟地址=16-l1

所以
l0=2,l1=2

b0和b1有很多种可行组合，事实上只要4≤b1-b0≤128均可。

答案验证

```
python2 segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0
0 --l0 2 --b1 4 --l1 2 -c
ARG seed 0
ARG address space size 16
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x00000004 (decimal 4)
  Segment 1 limit                  : 2

Virtual Address Trace
  VA  0: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:    0)
  VA  1: 0x00000001 (decimal:    1) --> VALID in SEG0: 0x00000001 (decimal:    1)
  VA  2: 0x00000002 (decimal:    2) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000003 (decimal:    3) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
  VA  5: 0x00000005 (decimal:    5) --> SEGMENTATION VIOLATION (SEG0)
  VA  6: 0x00000006 (decimal:    6) --> SEGMENTATION VIOLATION (SEG0)
  VA  7: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  8: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  9: 0x00000009 (decimal:    9) --> SEGMENTATION VIOLATION (SEG1)
  VA 10: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA 11: 0x0000000b (decimal:   11) --> SEGMENTATION VIOLATION (SEG1)
  VA 12: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA 13: 0x0000000d (decimal:   13) --> SEGMENTATION VIOLATION (SEG1)
  VA 14: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x00000002 (decimal:    2)
  VA 15: 0x0000000f (decimal:   15) --> VALID in SEG1: 0x00000003 (decimal:    3)
```

```
python2 segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0
0 --l0 2 --b1 128 --l1 2 -c
ARG seed 0
ARG address space size 16
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x00000080 (decimal 128)
  Segment 1 limit                  : 2

Virtual Address Trace
  VA  0: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:    0)
  VA  1: 0x00000001 (decimal:    1) --> VALID in SEG0: 0x00000001 (decimal:    1)
  VA  2: 0x00000002 (decimal:    2) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000003 (decimal:    3) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
  VA  5: 0x00000005 (decimal:    5) --> SEGMENTATION VIOLATION (SEG0)
  VA  6: 0x00000006 (decimal:    6) --> SEGMENTATION VIOLATION (SEG0)
  VA  7: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  8: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  9: 0x00000009 (decimal:    9) --> SEGMENTATION VIOLATION (SEG1)
  VA 10: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA 11: 0x0000000b (decimal:   11) --> SEGMENTATION VIOLATION (SEG1)
  VA 12: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA 13: 0x0000000d (decimal:   13) --> SEGMENTATION VIOLATION (SEG1)
  VA 14: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x0000007e (decimal:  126)
  VA 15: 0x0000000f (decimal:   15) --> VALID in SEG1: 0x0000007f (decimal:  127)
```

经验证，结果正确。