16

Q 1

使用指令为

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

分别看到这些指令在终端下的输出

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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.
```

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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.
```

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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.
```

第一题分析

利用`-h`指令查看指令参数的意义.

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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
```

那么指令`segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0`.的真实意义:

设置:

- `-a 128` 地址空间大小为 128
- `-p 512` 物理内存大小为 512
- `-b 0` 段0基址寄存器的值为 0
- `-l 20` 段0基址寄存器的限制值为 20
- `-B 512` 段1基址寄存器的值为 512
- `-L 20` 段1基址寄存器的限制值为 20
- `-s 0` 随机种子为 0

根据这些就完全知道了整个地址翻译过程的全部信息.

如果不考虑分段的情况的话,其实解析的过程和之前第15章的第一个习题是一致的.

但是现在有两个段.看到终端输出关于描述段的内容中, 有这样的描述:

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

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

这里有两句话:`grows positive`,`grows negative`. 这两句话是分别描述段0和段1的, 说明段0是正向增长的,但是段1是反向增长的.

具体地址转换

因为这个题目要做的地址转换很多,但是都是相同的分析方法, 所以就以第一条指令的第一个待转换的地址为例子分析.

```
0x0000006c (decimal:  108)
```

如果想知道如何转换,首先要知道这个虚拟地址应该在哪个段. 但是题目和`-h`指令并没有描述如何找虚拟地址的分段的. 所以我找了一下`README-segmentation`文件,发现果然有一段描述是关于如何找如何分段的:

```
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.
```

这段话的主要意思就是, 给你一个虚拟地址`108`之后,把他转化为二进制,看最高位是`0`还是`1`. 但是如何确定最高位是哪一位呢?

因为这里所给出来的地址空间的大小为`128`即从0~(2^8-1).也就是说最高位是第7位.所以`108`转换为二进制之后是`1101100`,说明这个虚拟地址分配给了段1.但是其实知道了是观察最高位之后,其实就不用这么麻烦地每次转化为二进制看最高位是不是1. 直接比较虚拟地址和2^7也就是64的大小关系就可以. 得到一个虚拟地址和段分配的映射函数`Seg(decimal)`

$$Seg(decimal) = \begin{cases} 0 & decimal < 64 \\ 1 & decimal \geq 64 \end{cases}$$

因为当时段0的时候是正向增长,而段1的时候是反向增长.根据15章内容知道正向增长的计算公式`base`+`decimal`, 而反向增长的时候稍微复杂一点,是`base`-(`asize`-`decimal`).就是基地址减去存储空间和虚拟地址的差.带入上一个映射函数可以得到虚拟地址到最终的地址的映射函数`Decimal(decimal)`

$$Decimal(decimal) = \begin{cases} base + decimal & decimal < 64 \\ base - (asize - decimal) & decimal \geq 64 \end{cases}$$

整理可得

$$Decimal(decimal) = \begin{cases} base + decimal & decimal < 64 \\ base + decimal - asize & decimal \geq 64 \end{cases}$$

看到表达式中,其实就是如果`decimal`大于64的时候要多减一个`asize`.最后还要判断是否是合法的, 合法的就是用虚拟地址求出来的相对地址小于`limit`

由于题目众多, 为了得到结果, 直接编程实现更方便. 一下为c++核心代码:

```
bool legalTest(int segment,int decimal){
    if (segment)
        return (segment-limit) <= decimal;
    else
        return (segment+limit) > decimal
}

segment = Seg(decimal);
decimal < 64 ? decimal -= asize : ;
decimal += base;
Legal = legalTest(segment,decimal);
```

得到的三个指令的结果分别为:

```
0: 0x0000006c (decimal:  108) : 界内 在 段1: 0x000001ec (decimal:  492)
1: 0x00000061 (decimal:   97) : 界外 (段1)
2: 0x00000035 (decimal:   53) : 界外 (段0)
3: 0x00000021 (decimal:   33) : 界外 (段0)
4: 0x00000041 (decimal:   65) : 界外 (段1)
```

```
0: 0x00000011 (decimal:   17) : 界内 在 段0: 0x00000011 (decimal:   17)
1: 0x0000006c (decimal:  108) : 界内 在 段1: 0x000001ec (decimal:  492)
2: 0x00000061 (decimal:   97) : 界外 (段1)
3: 0x00000020 (decimal:   32) : 界外 (段0)
4: 0x0000003f (decimal:   63) : 界外 (段0)
```

```
0: 0x0000007a (decimal:  122) : 界内 在 段1: 0x000001fa (decimal:  506)
1: 0x00000079 (decimal:  121) : 界内 在 段1: 0x000001f9 (decimal:  505)
2: 0x00000007 (decimal:    7) : 界内 在 段0: 0x00000007 (decimal:    7)
3: 0x0000000a (decimal:   10) : 界内 在 段0: 0x0000000a (decimal:   10)
4: 0x0000006a (decimal:  106) : 界外 (段1)
```

答案1

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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: 0x000001ec (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)
```

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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)
```

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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)
```

对比答案, 完全一致. 说明我的理解和推论都正确

Q 2

第二题分析

本题的基本信息全部建立在上一道题目的假设基础上. 因为随机基本信息的设置基本一致, 随机种子只是又不同的待翻译地址而已, 所以就以`segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0`这个指令为例子分析此题目.

因为第一个问得到的映射函数,这个题的前四个问直接带入求解即可.

$$Decimal(decimal) = \begin{cases} base + decimal & decimal < 64 \\ base + decimal - asize & decimal \geq 64 \end{cases}$$

段 0 中最高的合法虚拟地址是

$$asize + limit = 0 + (20 - 1) = 19$$

这里要减这个1,因为计算机从0开始. 同理, 段 1 中最低的合法虚拟地址是

$$asize - limit = 128 - 20 = 108$$

最低非法地址就是

$$19 + 1 = 20$$

最高非法地址就是

$$108 - 1 = 107$$

答案2

最终使用指令`./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1 -A 19,108,20,107 -c`验证推论

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ python2 ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1 -A 19,108,20,107 -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: 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)
```

发现完全一致, 恰好19、108可以访问, 20、107访问不了

Q 3

分析 3

在中文教材中是这样描述的

> 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，违反，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 ?`

题目是有问题的,因为题目描述了16个待转化地址,结果题目描述的这些地址的合法性只有"有效，有效，违规，违反，有效，有效"这六个.所以出错了.

那么看英文原教材是这样描述题目的:

> Let’s say we have a tiny 16-byte address space in a 128-byte physical memory. What base and bounds would you set up so as to get the simulator to generate the following translation results for the specified address stream: valid, valid, violation, ..., violation, valid, valid? Assume the following parameters:`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 ?`

这段描述中说地址的合法性是:"valid, valid, violation, ..., violation, valid, valid".也就是只有前两个和后两个是合法的.其余的都是非法的.应该是翻译的时候没有把这个省略号翻译进来.

先看这个指令会给出什么样的待翻译地址:

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ python2 ./segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
ARG seed 0
ARG address space size 16
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x00000035 (decimal 53)
  Segment 0 limit                  : 7

  Segment 1 base  (grows negative) : 0x00000028 (decimal 40)
  Segment 1 limit                  : 7

Virtual Address Trace
  VA  0: 0x00000000 (decimal:    0) --> PA or segmentation violation?
  VA  1: 0x00000001 (decimal:    1) --> PA or segmentation violation?
  VA  2: 0x00000002 (decimal:    2) --> PA or segmentation violation?
  VA  3: 0x00000003 (decimal:    3) --> PA or segmentation violation?
  VA  4: 0x00000004 (decimal:    4) --> PA or segmentation violation?
  VA  5: 0x00000005 (decimal:    5) --> PA or segmentation violation?
  VA  6: 0x00000006 (decimal:    6) --> PA or segmentation violation?
  VA  7: 0x00000007 (decimal:    7) --> PA or segmentation violation?
```

```
  VA  8: 0x00000008 (decimal:    8) --> PA or segmentation violation?
  VA  9: 0x00000009 (decimal:    9) --> PA or segmentation violation?
  VA 10: 0x0000000a (decimal:   10) --> PA or segmentation violation?
  VA 11: 0x0000000b (decimal:   11) --> PA or segmentation violation?
  VA 12: 0x0000000c (decimal:   12) --> PA or segmentation violation?
  VA 13: 0x0000000d (decimal:   13) --> PA or segmentation violation?
  VA 14: 0x0000000e (decimal:   14) --> PA or segmentation violation?
  VA 15: 0x0000000f (decimal:   15) --> 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.
```

就是让虚拟地址0、1和14、15在界内,其余的在界外.

根据问题1可知,这里的两个limit都是2,所以推导出

- 段0的`base`为0
- 段0的`limit`为2
- 段1的`base`为16
- 段1的`limit`为2

得到查看答案的指令:`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 16 --l1 2 -c`

答案

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter16/HW-Segmentation$ 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 16 --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) : 0x00000010 (decimal 16)
  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: 0x0000000e (decimal:   14)
VA 15: 0x0000000f (decimal:   15) --> VALID in SEG1: 0x0000000f (decimal:   15)
```

看到答案中实际上也是只有前两个和最后两个在界内,符合推论,结果正确.