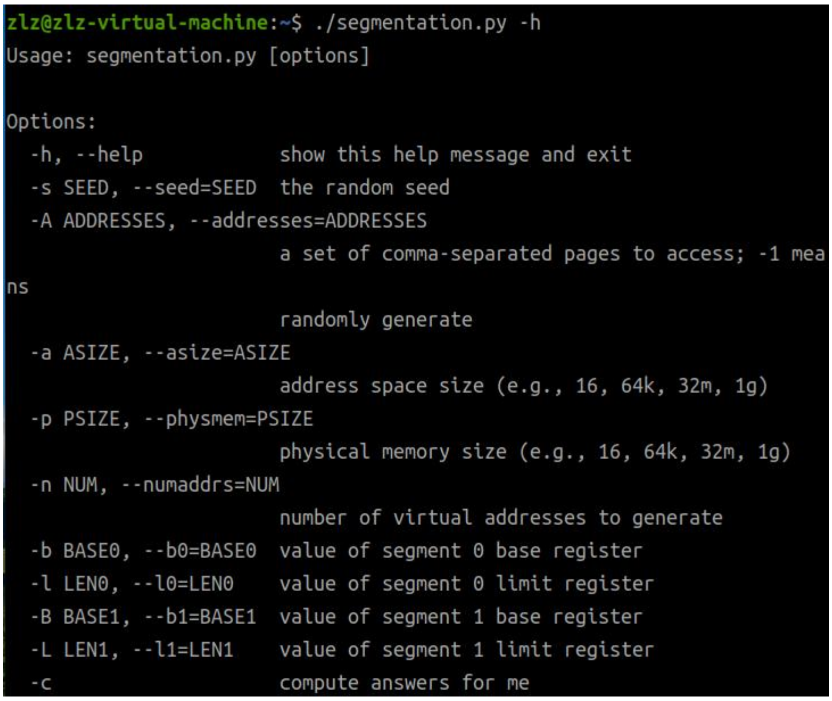
第十六章

1.阅读-h 文件



```
zlz@zlz-virtual-machine:~$ ./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 mea
ns
                        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
```

-a 指定地址空间大小

-p 指定物理内存大小

-b 指定段 0 的基址寄存器的值

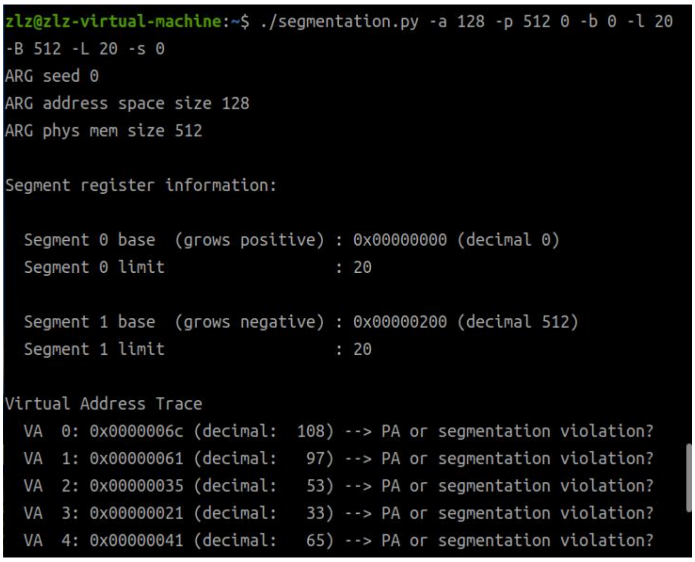
-l 指定段 0 的限长寄存器的值

-B 指定段 1 的基址寄存器的值

-L 指定段 1 的限长寄存器的值

-s 指定随机数种子

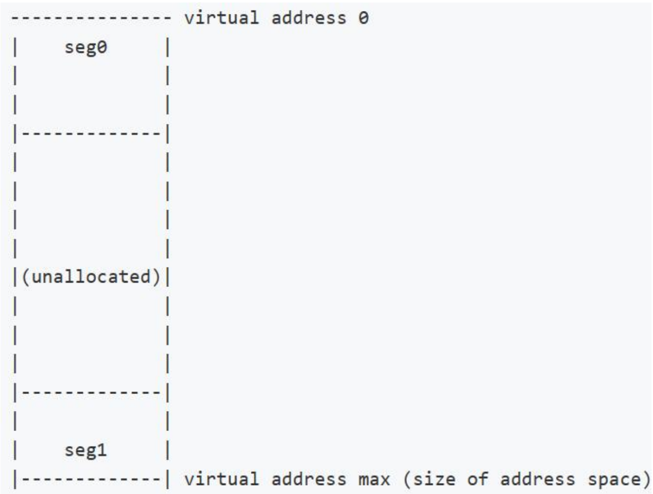
（1）./segmentation.py -a 128 -p 512 0 -b 0 -l 20 -B 512 -L 20 -s 0



```
zlz@zlz-virtual-machine:~$ ./segmentation.py -a 128 -p 512 0 -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?
```

虚拟地址大小为 128，物理地址空间大小 512

其中，物理地址空间被划分为两段，如下图所示的分布。



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

虚拟地址的顶部位决定了地址所在的段，0 表示段 0，1 表示段 1，段 0 向正方向增长（朝向更高的地址），而段 1 向负方向增长。

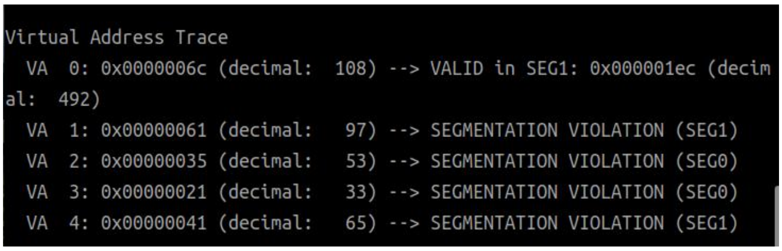
VA 0: 0x0000006c (decimal:108)110 1100 OFFSET=-20 -->VALID in SEG1: 0x000001ec(decimal:492)

VA 1: 0x00000061 (decimal: 97)110 0001 OFFSET=-31-->SEGMENTATION VIOLATION (SEG1)

VA 2: 0x00000035 (decimal: 53)011 0101 OFFSET=53--> SEGMENTATION VIOLATION (SEG0)

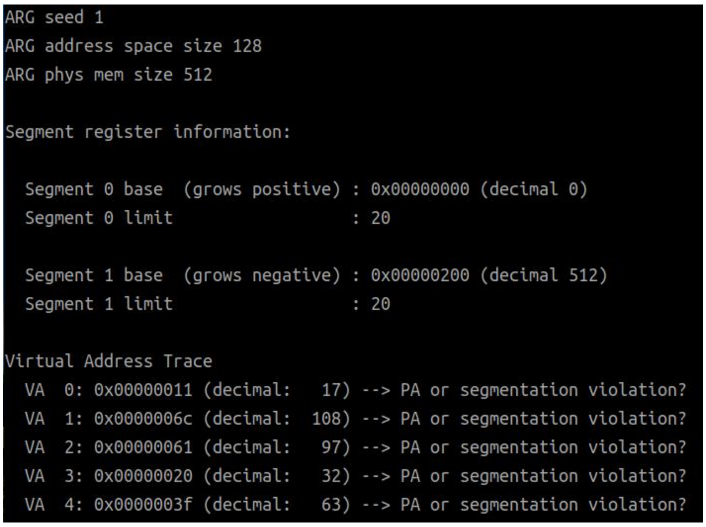
VA 3: 0x00000021 (decimal: 33)010 0001 OFFSET=33--> SEGMENTATION VIOLATION (SEG0)

VA 4: 0x00000041 (decimal: 65)100 0001 OFFSET=-63--> SEGMENTATION VIOLATION (SEG1)



```
Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decim
al:  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)
```

(2)./segmentation.py -a 128 -p 512 0 -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?
```

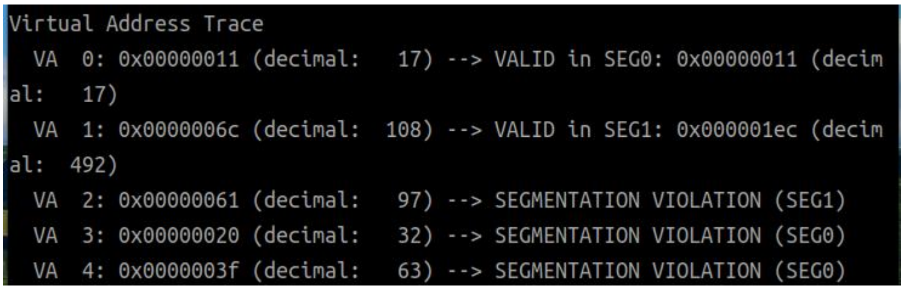
VA 0: 0x00000011 (decimal:17)001 0001 OFFSET=17-->VALID in SEG0: 0x00000011 (decimal: 17)

VA 1: 0x0000006c (decimal:108)110 1100 OFFSET=-20-->VALID in SEG1: 0x000001ec (decimal: 492)

VA 2: 0x00000061 (decimal:97)110 0001 OFFSET=-31--> SEGMENTATION VIOLATION (SEG1)

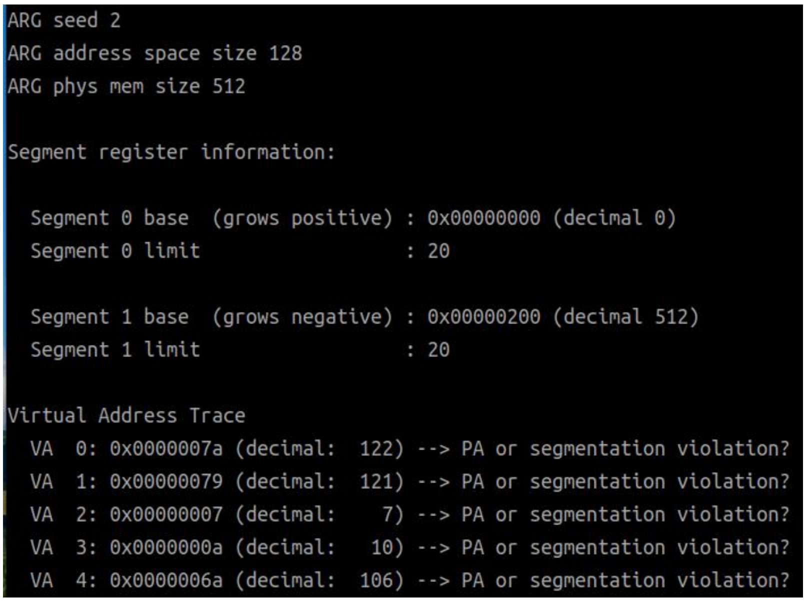
VA 3: 0x00000020 (decimal:32)010 0000 OFFSET=32--> SEGMENTATION VIOLATION (SEG0)

VA 4: 0x0000003f (decimal:63)011 1111 OFFSET=63--> SEGMENTATION VIOLATION (SEG0)



```
Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> VALID in SEG0: 0x00000011 (decim
al:   17)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decim
al:  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)
```

(3)./segmentation.py -a 128 -p 512 0 -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?
```

VA 0: 0x0000007a (decimal:122)111 1010 OFFSET=-6-->VALID in SEG1: 0x000001fa (decimal: 506)

VA 1: 0x00000079 (decimal:121)111 1001 OFFSET=-7-->VALID in SEG1: 0x000001f9 (decimal: 505)

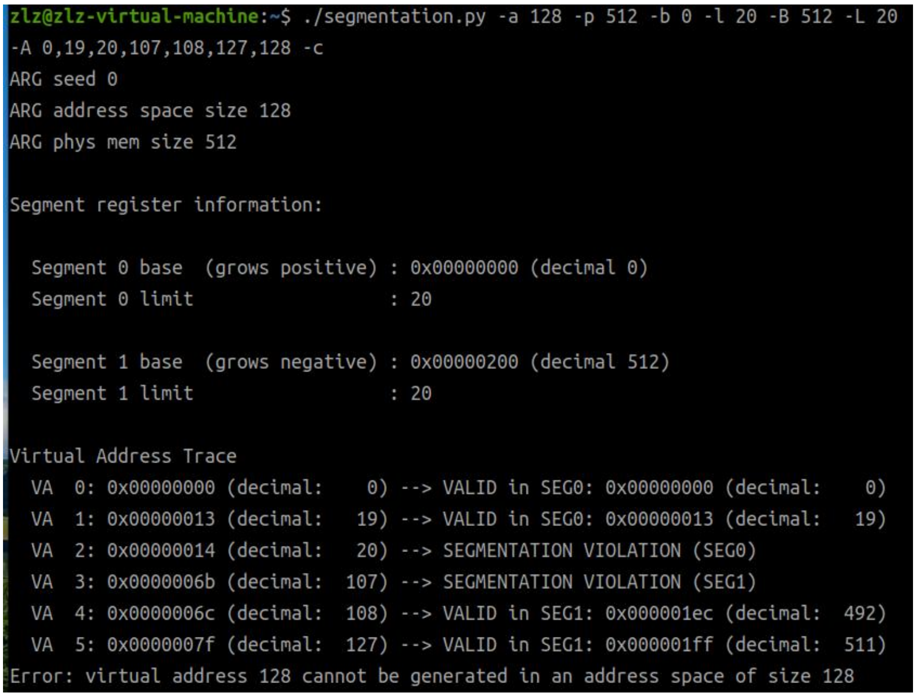
VA 2: 0x00000007 (decimal:7)000 0111 OFFSET=7-->VALID in SEG0: 0x00000007 (decimal: 7)

VA 3: 0x0000000a (decimal:10)000 1010 OFFSET=10-->VALID in SEG0: 0x0000000a (decimal: 10)

VA 4: 0x0000006a (decimal:106)110 1010 OFFSET=-22-->SEGMENTATION VIOLATION (SEG1)

2.段0中最高的合法虚拟地址是0x00000013(decimal:19)
段1中最低的合法虚拟地址是0x0000006c（decimal:108)
整个地址空间中最低的非法地址是0x00000014(decimal:20)
整个地址空间中最高的非法地址是0x0000006b(decimal:107)

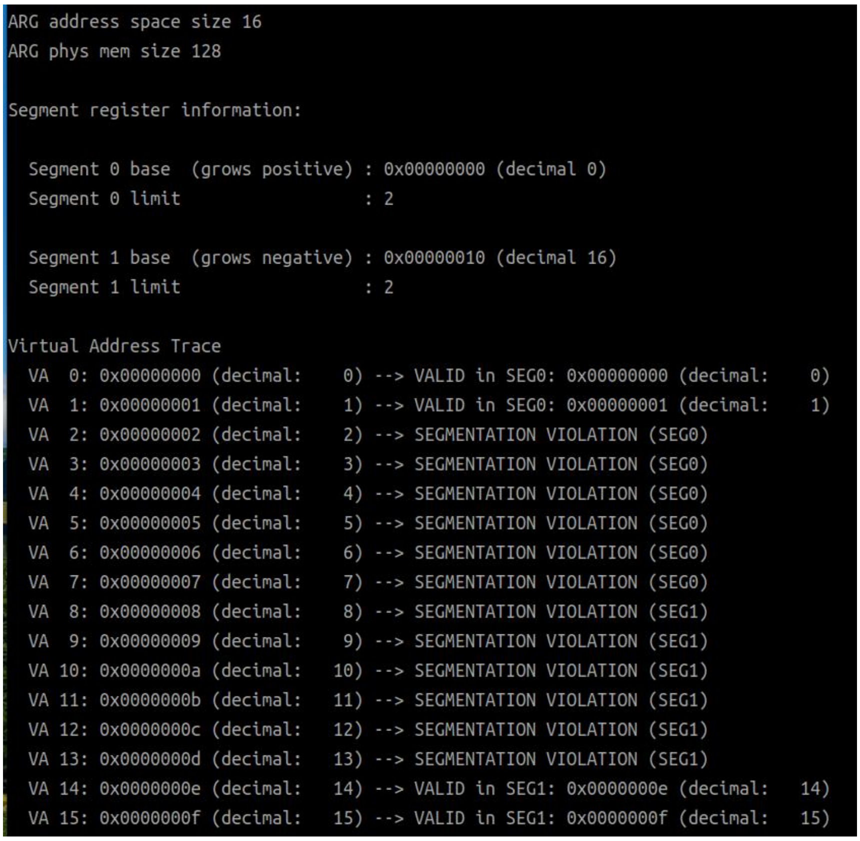
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -A 19,20,107,108,12 -c



```
zlz@zlz-virtual-machine:~$ ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20
-A 0,19,20,107,108,127,128 -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: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:    0)
  VA  1: 0x00000013 (decimal:   19) --> VALID in SEG0: 0x00000013 (decimal:   19)
  VA  2: 0x00000014 (decimal:   20) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x0000006b (decimal:  107) --> SEGMENTATION VIOLATION (SEG1)
  VA  4: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  5: 0x0000007f (decimal:  127) --> VALID in SEG1: 0x000001ff (decimal:  511)
Error: virtual address 128 cannot be generated in an address space of size 128
```

根据显示，结果符合我们的预期

3.根据题目显示的只有前两个和最后两个是有效的，即虚拟地址0，1和14，15可以转换成功，需要保证这两个段长度为2，所以b0=0,b1=16,l0=2,l1=2。
所以应该设置为
./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

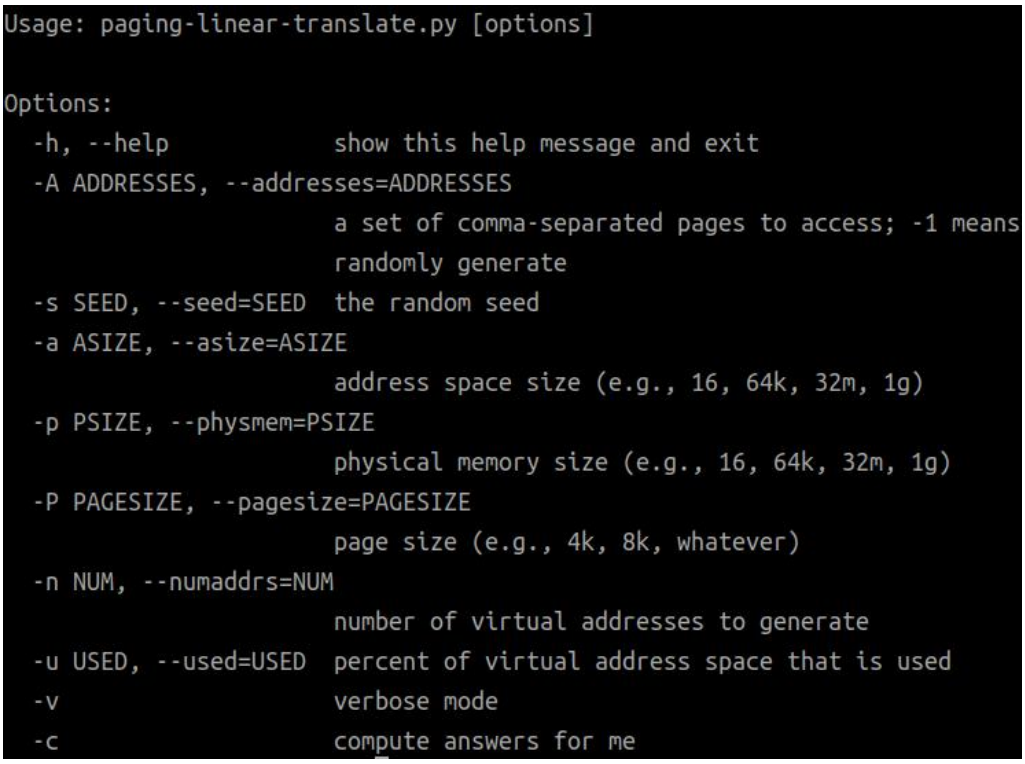


```
ARG address space size 16
ARG phys mem size 128

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)
```

结果符合预期。

第十八章

1.阅读-h 文件



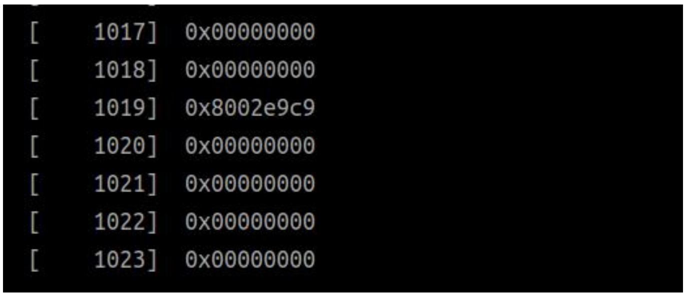
```
Usage: paging-linear-translate.py [options]

Options:
  -h, --help            show this help message and exit
  -A ADDRESSES, --addresses=ADDRESSES
                        a set of comma-separated pages to access; -1 means
                        randomly generate
  -s SEED, --seed=SEED  the random seed
  -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)
  -P PAGESIZE, --pagesize=PAGESIZE
                        page size (e.g., 4k, 8k, whatever)
  -n NUM, --numaddrs=NUM
                        number of virtual addresses to generate
  -u USED, --used=USED  percent of virtual address space that is used
  -v                    verbose mode
  -c                    compute answers for me
```

-s 设置随机数种子
-a 指定地址空间大小
-p 指定物理内存大小
-P 设置页面大小
-n NUM 设置要生成的虚拟地址数量
-u 设置虚拟地址空间被使用的百分比

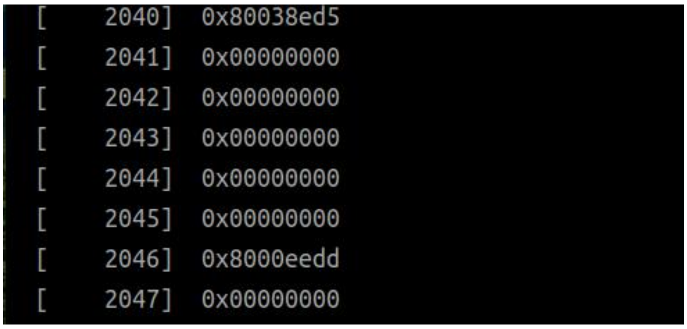
（1）

```
./paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
./paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0
./paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0
```

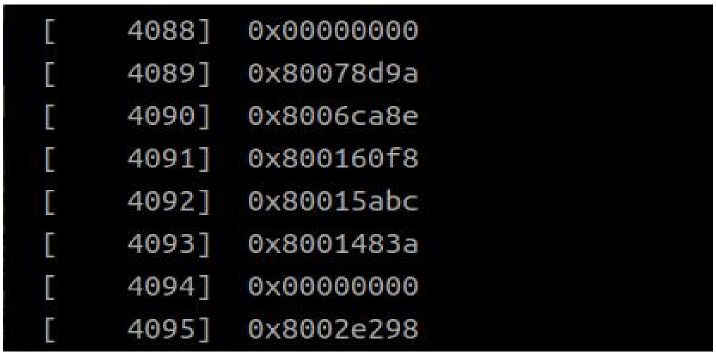
其中，每次增加地址空间大小，对应页表大小设置为 1kb，地址空间分别为 1mb，2mb，4mb，物理地址空间大小为 512mb，页表项分别为 1mb/1kb=1024,2mb/1kb=2048,4mb/1kb=4096,如果假定一个页表需要 4 字节，那么页表项大小分别为 4kb，8kb，12kb。在页大小相同的情况下，地址空间增大，页表项随之增长，页表增大。



```
[     1017]  0x00000000
[     1018]  0x00000000
[     1019]  0x8002e9c9
[     1020]  0x00000000
[     1021]  0x00000000
[     1022]  0x00000000
[     1023]  0x00000000
```



```
[     2040]  0x80038ed5
[     2041]  0x00000000
[     2042]  0x00000000
[     2043]  0x00000000
[     2044]  0x00000000
[     2045]  0x00000000
[     2046]  0x8000eedd
[     2047]  0x00000000
```



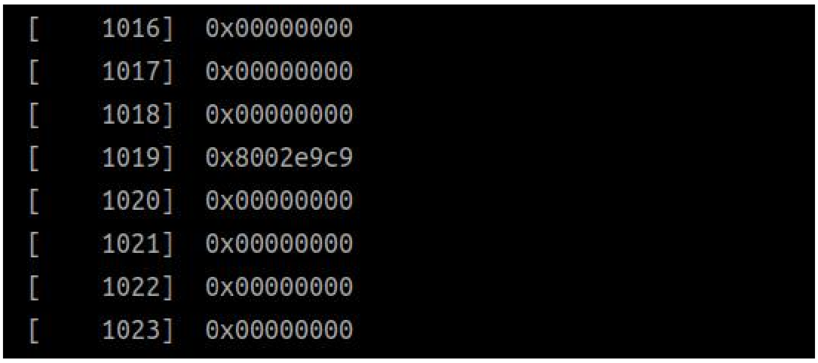
```
[     4088]  0x00000000
[     4089]  0x80078d9a
[     4090]  0x8006ca8e
[     4091]  0x800160f8
[     4092]  0x80015abc
[     4093]  0x8001483a
[     4094]  0x00000000
[     4095]  0x8002e298
```

和我们所预期的结果一致

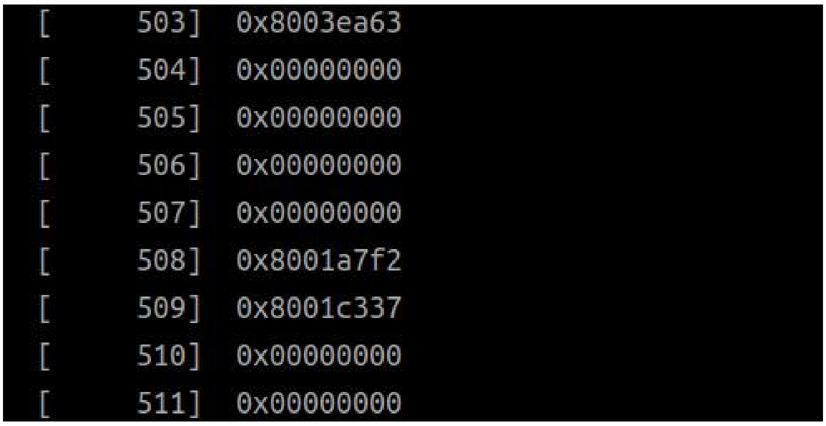
（2）

```
./paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
./paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
./paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0
```

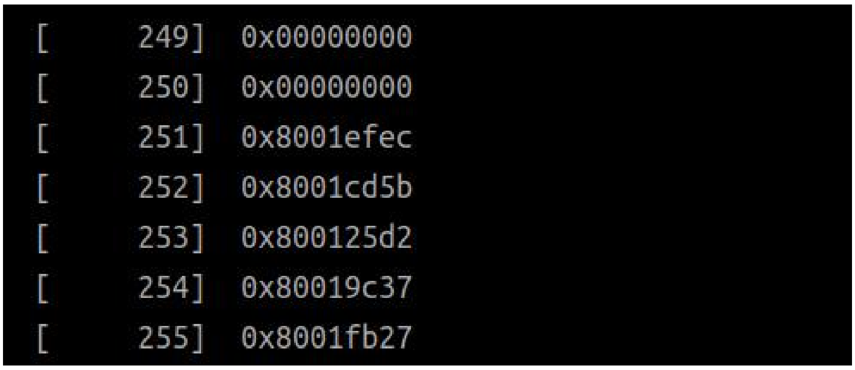
其中，每次增加页面的大小，对应的大小分别为 1kb,2kb,4kb，地址空间一致，为 1mb，物理地址空间大小为 512mb ，页表项分别为 1mb/1kb=1024,1mb/2kb=512,1mb/4kb=256，假设每个页表项需要 4 字节的空间，那么页表项大小分别为 4kb,2kb,1kb。在地址空间大小相同的情况下，页的大小增大，页表项减少，页表减小。



```
[    1016]  0x00000000
[    1017]  0x00000000
[    1018]  0x00000000
[    1019]  0x8002e9c9
[    1020]  0x00000000
[    1021]  0x00000000
[    1022]  0x00000000
[    1023]  0x00000000
```



```
[     503]  0x8003ea63
[     504]  0x00000000
[     505]  0x00000000
[     506]  0x00000000
[     507]  0x00000000
[     508]  0x8001a7f2
[     509]  0x8001c337
[     510]  0x00000000
[     511]  0x00000000
```



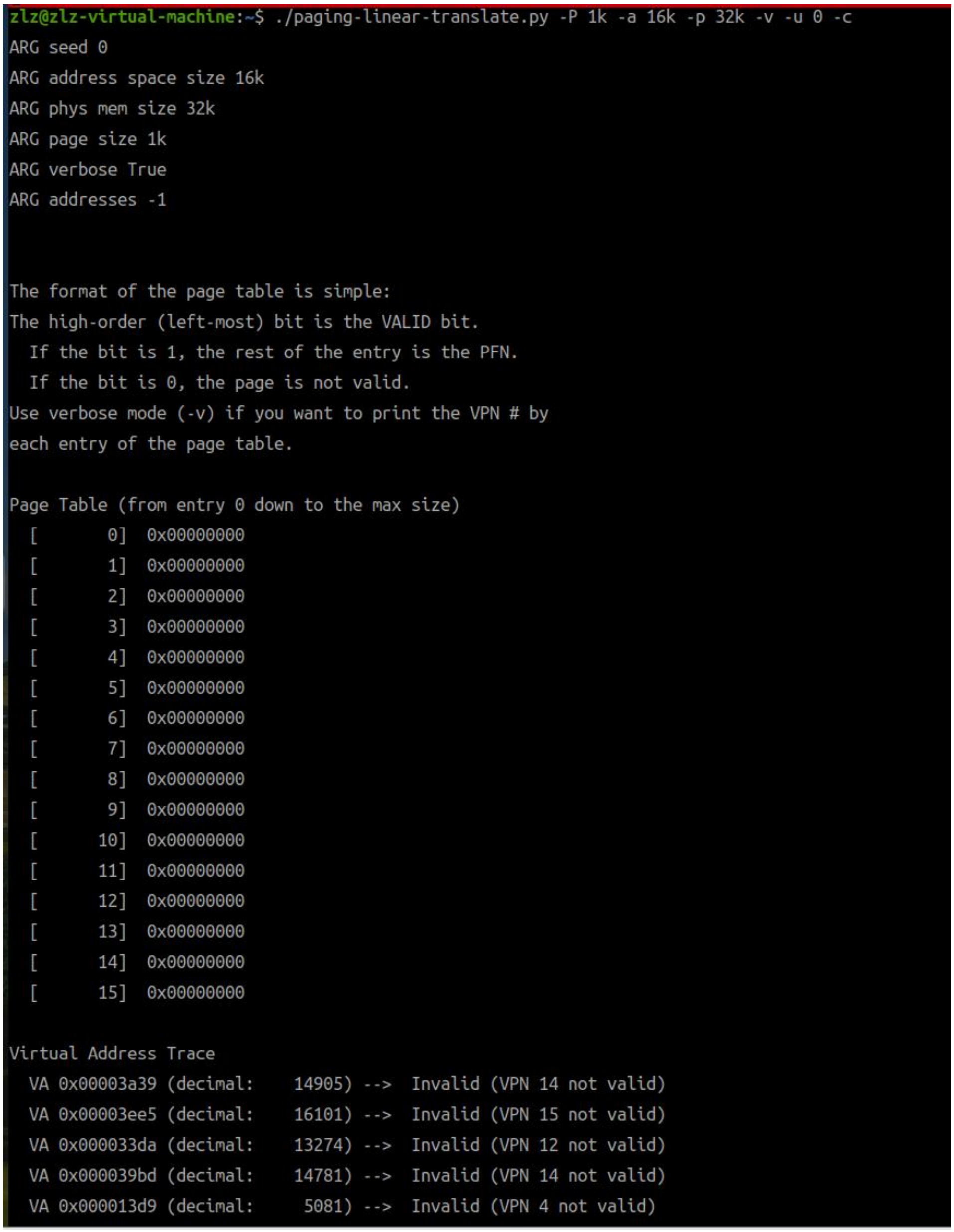
```
[     249]  0x00000000
[     250]  0x00000000
[     251]  0x8001efec
[     252]  0x8001cd5b
[     253]  0x800125d2
[     254]  0x80019c37
[     255]  0x8001fb27
```

和我们所预期的结果一致

不应该使用很大的页，因为过于大的页会产生很多的内部碎片，造成浪费。

2.

```
./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0 -c
```

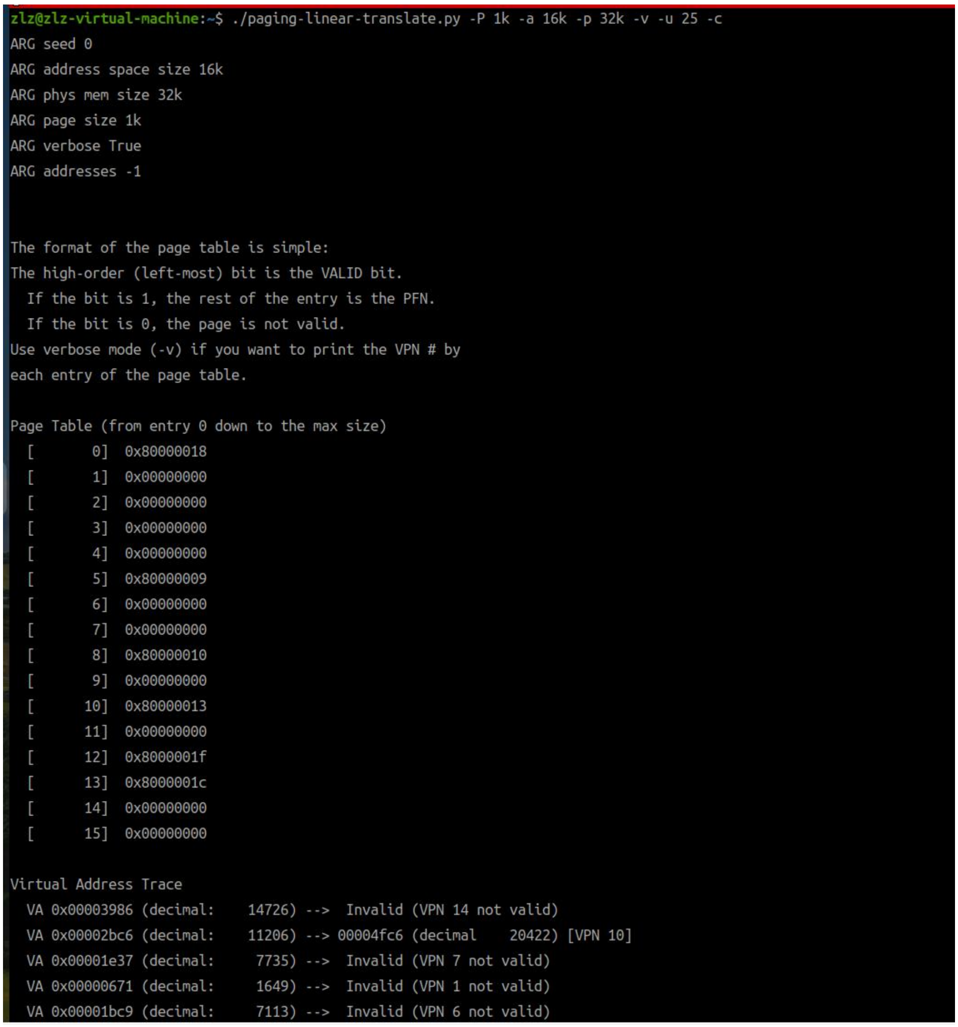


```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0 -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x00000000
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x00000000
  [       4]  0x00000000
  [       5]  0x00000000
  [       6]  0x00000000
  [       7]  0x00000000
  [       8]  0x00000000
  [       9]  0x00000000
  [      10]  0x00000000
  [      11]  0x00000000
  [      12]  0x00000000
  [      13]  0x00000000
  [      14]  0x00000000
  [      15]  0x00000000

Virtual Address Trace
  VA 0x00003a39 (decimal:    14905) --> Invalid (VPN 14 not valid)
  VA 0x00003ee5 (decimal:    16101) --> Invalid (VPN 15 not valid)
  VA 0x000033da (decimal:    13274) --> Invalid (VPN 12 not valid)
  VA 0x000039bd (decimal:    14781) --> Invalid (VPN 14 not valid)
  VA 0x000013d9 (decimal:     5081) --> Invalid (VPN 4 not valid)
```

模拟程序中的几个虚拟地址均未转换成功，这是因为虚拟地址对应的 VPN 在地址空间中未分配页，无法查找到 PTE，所以不能成功。

```
Virtual Address Trace
  VA 0x00003a39 (decimal:    14905) --> Invalid (VPN 14 not valid)
  VA 0x00003ee5 (decimal:    16101) --> Invalid (VPN 15 not valid)
  VA 0x000033da (decimal:    13274) --> Invalid (VPN 12 not valid)
  VA 0x000039bd (decimal:    14781) --> Invalid (VPN 14 not valid)
  VA 0x000013d9 (decimal:     5081) --> Invalid (VPN 4 not valid)
```

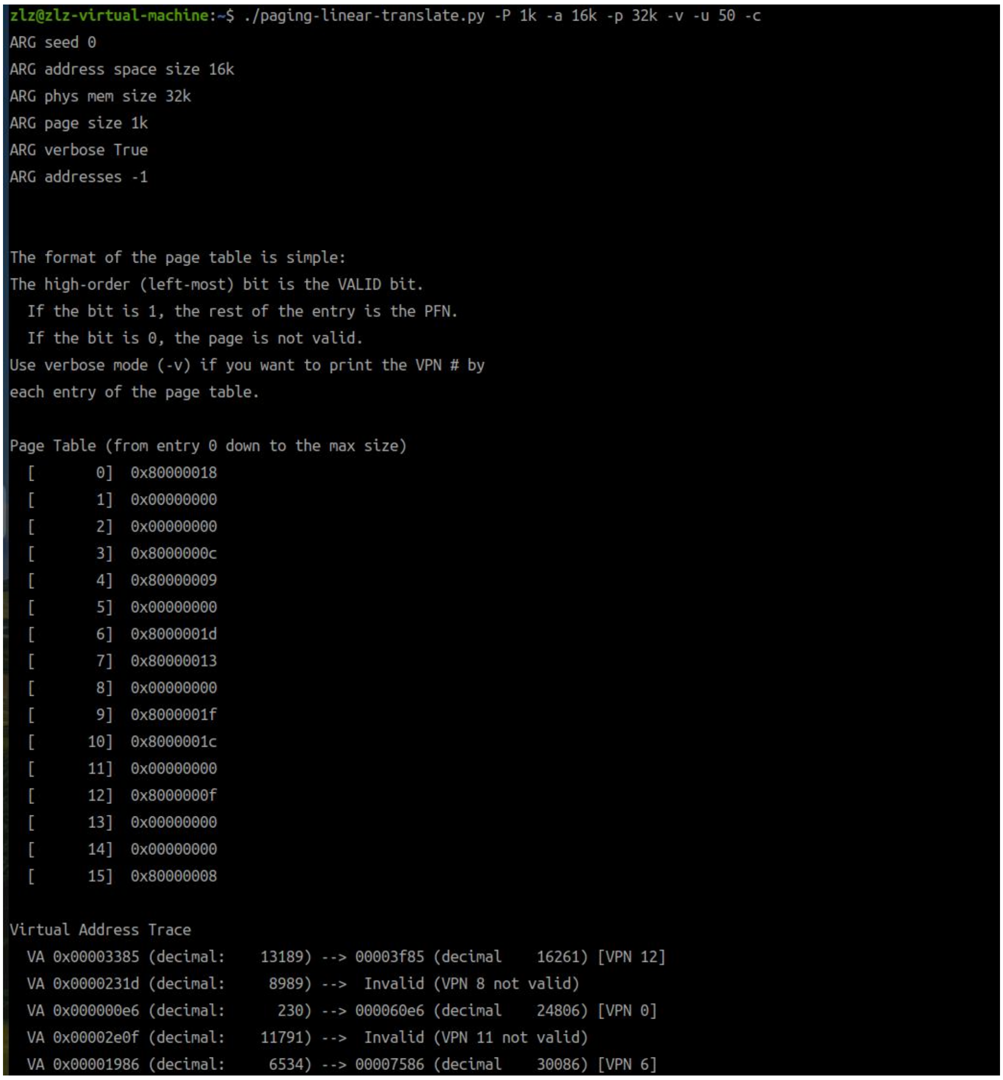
```
./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25 -c
```



```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25 -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x00000000
  [       4]  0x00000000
  [       5]  0x80000009
  [       6]  0x00000000
  [       7]  0x00000000
  [       8]  0x80000010
  [       9]  0x00000000
  [      10]  0x80000013
  [      11]  0x00000000
  [      12]  0x8000001f
  [      13]  0x8000001c
  [      14]  0x00000000
  [      15]  0x00000000

Virtual Address Trace
  VA 0x00003986 (decimal:    14726) -->  Invalid (VPN 14 not valid)
  VA 0x00002bc6 (decimal:    11206) --> 00004fc6 (decimal    20422) [VPN 10]
  VA 0x00001e37 (decimal:     7735) -->  Invalid (VPN 7 not valid)
  VA 0x00000671 (decimal:     1649) -->  Invalid (VPN 1 not valid)
  VA 0x00001bc9 (decimal:     7113) -->  Invalid (VPN 6 not valid)
```

```
Virtual Address Trace
  VA 0x00003986 (decimal:    14726) -->  Invalid (VPN 14 not valid)
  VA 0x00002bc6 (decimal:  11206) --> 00004fc6 (decimal   20422) [VPN 10]
  VA 0x00001e37 (decimal:     7735) -->  Invalid (VPN 7 not valid)
  VA 0x00000671 (decimal:     1649) -->  Invalid (VPN 1 not valid)
  VA 0x00001bc9 (decimal:     7113) -->  Invalid (VPN 6 not valid)
```

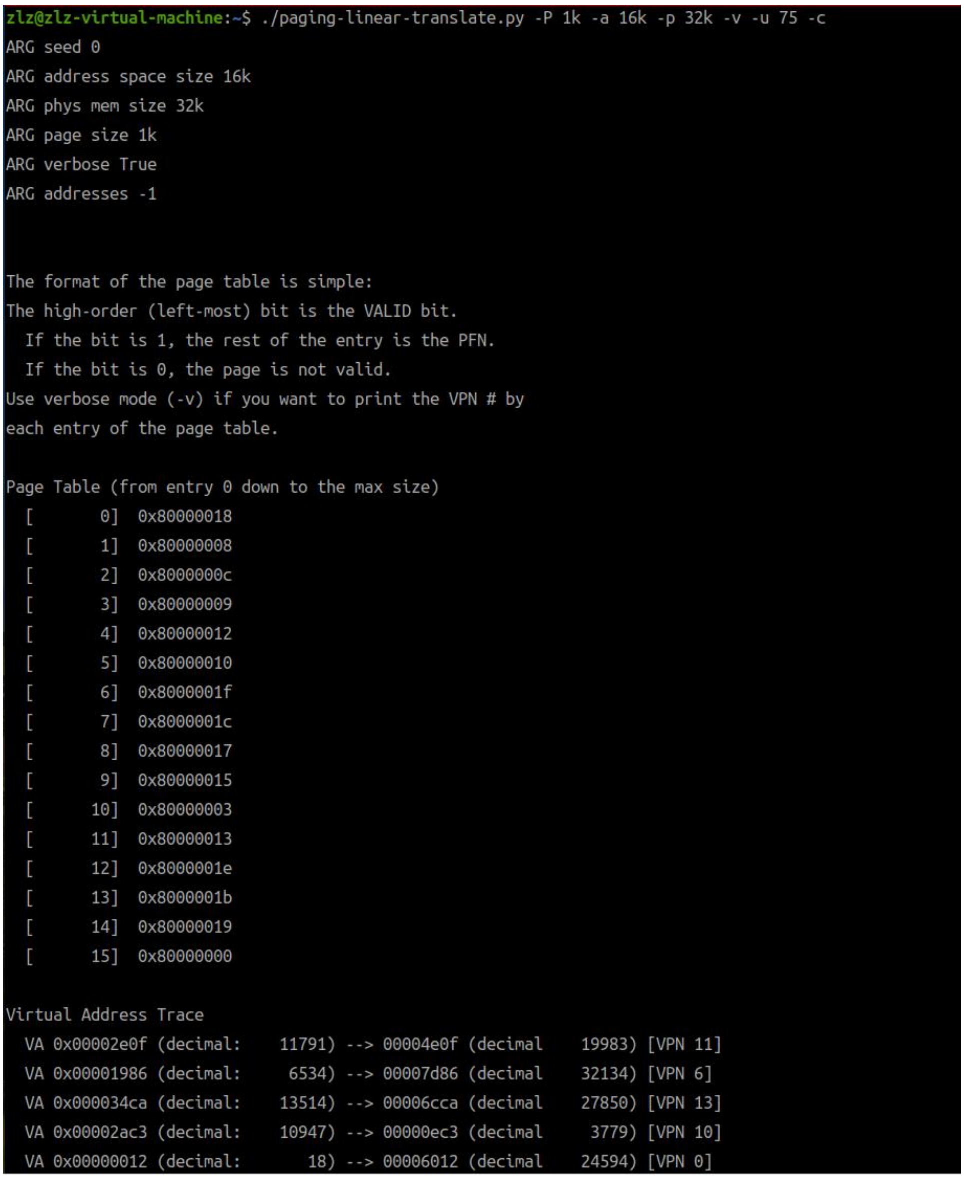
./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50 -c



```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50 -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x8000000c
  [       4]  0x80000009
  [       5]  0x00000000
  [       6]  0x8000001d
  [       7]  0x80000013
  [       8]  0x00000000
  [       9]  0x8000001f
  [      10]  0x8000001c
  [      11]  0x00000000
  [      12]  0x8000000f
  [      13]  0x00000000
  [      14]  0x00000000
  [      15]  0x80000008

Virtual Address Trace
  VA 0x00003385 (decimal:    13189) --> 00003f85 (decimal    16261) [VPN 12]
  VA 0x0000231d (decimal:     8989) -->  Invalid (VPN 8 not valid)
  VA 0x000000e6 (decimal:      230) --> 000060e6 (decimal    24806) [VPN 0]
  VA 0x00002e0f (decimal:    11791) -->  Invalid (VPN 11 not valid)
  VA 0x00001986 (decimal:     6534) --> 00007586 (decimal    30086) [VPN 6]
```

```
Virtual Address Trace
VA 0x00003385 (decimal:    13189) --> 00003f85 (decimal    16261) [VPN 12]
VA 0x0000231d (decimal:     8989) -->  Invalid (VPN 8 not valid)
VA 0x000000e6 (decimal:      230) --> 000060e6 (decimal    24806) [VPN 0]
VA 0x00002e0f (decimal:    11791) -->  Invalid (VPN 11 not valid)
VA 0x00001986 (decimal:     6534) --> 00007586 (decimal    30086) [VPN 6]
```

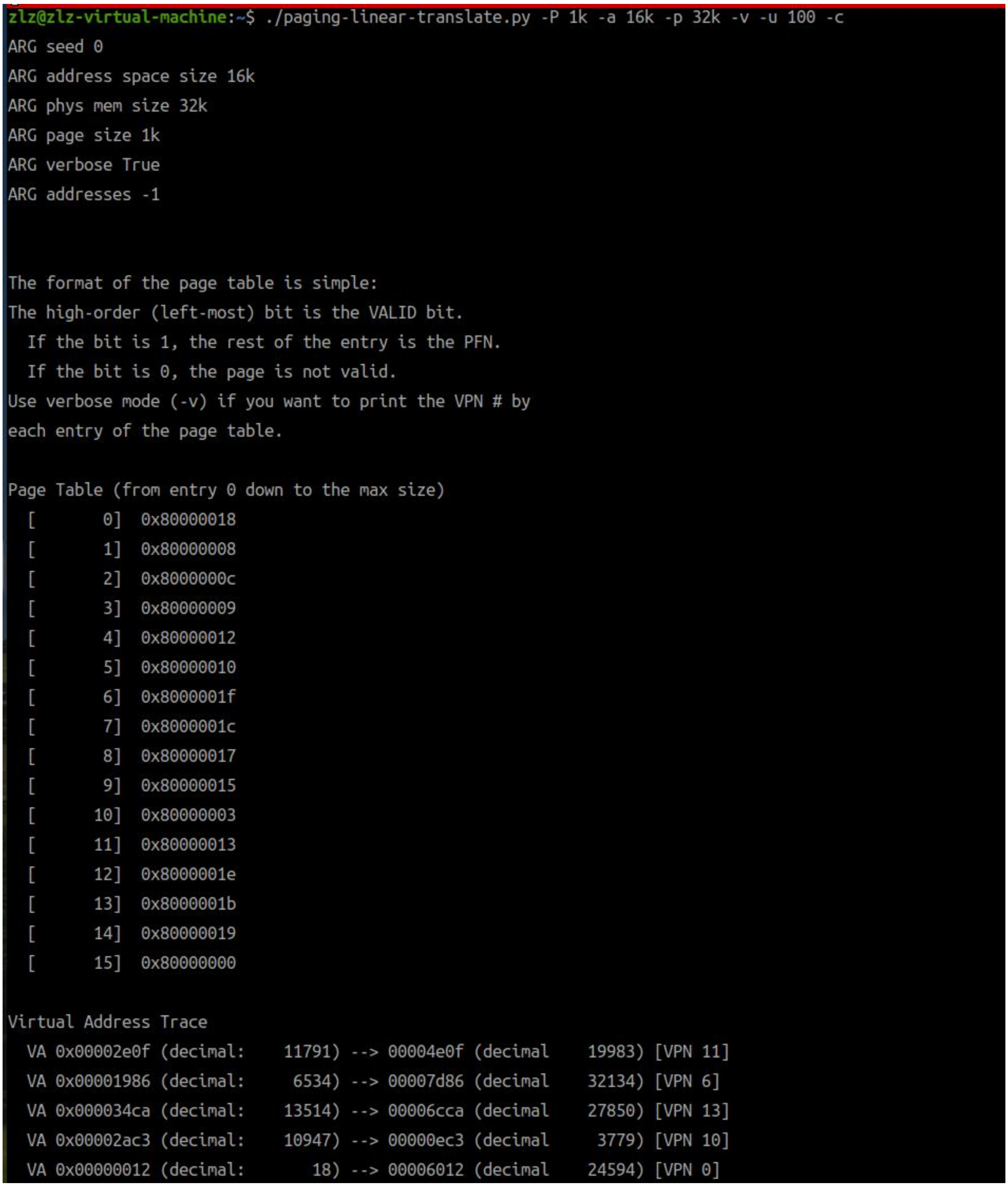
```
./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75 -c
```



```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75 -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x80000008
  [       2]  0x8000000c
  [       3]  0x80000009
  [       4]  0x80000012
  [       5]  0x80000010
  [       6]  0x8000001f
  [       7]  0x8000001c
  [       8]  0x80000017
  [       9]  0x80000015
  [      10]  0x80000003
  [      11]  0x80000013
  [      12]  0x8000001e
  [      13]  0x8000001b
  [      14]  0x80000019
  [      15]  0x80000000

Virtual Address Trace
  VA 0x00002e0f (decimal:    11791) --> 00004e0f (decimal    19983) [VPN 11]
  VA 0x00001986 (decimal:     6534) --> 00007d86 (decimal    32134) [VPN 6]
  VA 0x000034ca (decimal:    13514) --> 00006cca (decimal    27850) [VPN 13]
  VA 0x00002ac3 (decimal:    10947) --> 00000ec3 (decimal     3779) [VPN 10]
  VA 0x00000012 (decimal:       18) --> 00006012 (decimal    24594) [VPN 0]
```

```
Virtual Address Trace
VA 0x00002e0f (decimal:    11791) --> 00004e0f (decimal    19983) [VPN 11]
VA 0x00001986 (decimal:     6534) --> 00007d86 (decimal    32134) [VPN 6]
VA 0x000034ca (decimal:    13514) --> 00006cca (decimal    27850) [VPN 13]
VA 0x00002ac3 (decimal:    10947) --> 00000ec3 (decimal     3779) [VPN 10]
VA 0x00000012 (decimal:       18) --> 00006012 (decimal    24594) [VPN 0]
```

./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100 -c



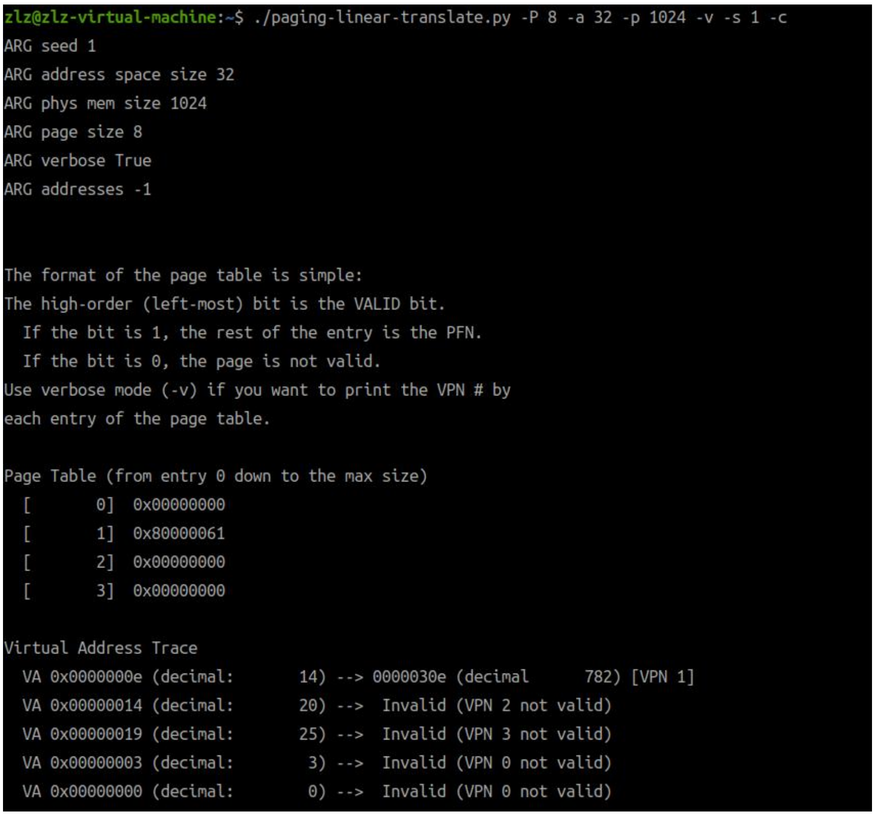
```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100 -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000018
  [       1]  0x80000008
  [       2]  0x8000000c
  [       3]  0x80000009
  [       4]  0x80000012
  [       5]  0x80000010
  [       6]  0x8000001f
  [       7]  0x8000001c
  [       8]  0x80000017
  [       9]  0x80000015
  [      10]  0x80000003
  [      11]  0x80000013
  [      12]  0x8000001e
  [      13]  0x8000001b
  [      14]  0x80000019
  [      15]  0x80000000

Virtual Address Trace
  VA 0x00002e0f (decimal:    11791) --> 00004e0f (decimal    19983) [VPN 11]
  VA 0x00001986 (decimal:     6534) --> 00007d86 (decimal    32134) [VPN 6]
  VA 0x000034ca (decimal:    13514) --> 00006cca (decimal    27850) [VPN 13]
  VA 0x00002ac3 (decimal:    10947) --> 00000ec3 (decimal     3779) [VPN 10]
  VA 0x00000012 (decimal:       18) --> 00006012 (decimal    24594) [VPN 0]
```

```
Virtual Address Trace
VA 0x00002e0f (decimal:    11791) --> 00004e0f (decimal    19983) [VPN 11]
VA 0x00001986 (decimal:     6534) --> 00007d86 (decimal    32134) [VPN 6]
VA 0x000034ca (decimal:    13514) --> 00006cca (decimal    27850) [VPN 13]
VA 0x00002ac3 (decimal:    10947) --> 00000ec3 (decimal     3779) [VPN 10]
VA 0x00000012 (decimal:       18) --> 00006012 (decimal    24594) [VPN 0]
```

可以得到结论，当每个地址空间中的页的百分比不断增加，有效地址也不断增多。

3.

./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1 -c



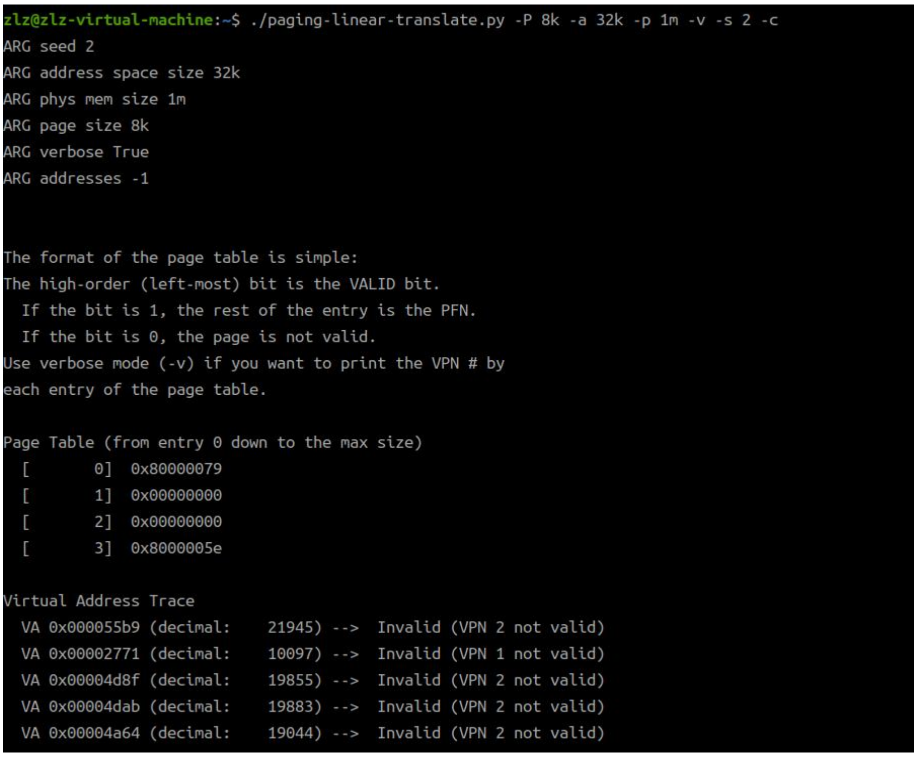
```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1 -c
ARG seed 1
ARG address space size 32
ARG phys mem size 1024
ARG page size 8
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x00000000
  [       1]  0x80000061
  [       2]  0x00000000
  [       3]  0x00000000

Virtual Address Trace
  VA 0x0000000e (decimal:       14) --> 0000030e (decimal      782) [VPN 1]
  VA 0x00000014 (decimal:       20) -->  Invalid (VPN 2 not valid)
  VA 0x00000019 (decimal:       25) -->  Invalid (VPN 3 not valid)
  VA 0x00000003 (decimal:        3) -->  Invalid (VPN 0 not valid)
  VA 0x00000000 (decimal:        0) -->  Invalid (VPN 0 not valid)
```

使用页面大小为8，地址空间大小为32，物理内存大小为1024，页数为4，设置随机种子为1

```
Virtual Address Trace
VA 0x0000000e (decimal:       14) --> 0000030e (decimal      782) [VPN 1]
VA 0x00000014 (decimal:       20) -->  Invalid (VPN 2 not valid)
VA 0x00000019 (decimal:       25) -->  Invalid (VPN 3 not valid)
VA 0x00000003 (decimal:        3) -->  Invalid (VPN 0 not valid)
VA 0x00000000 (decimal:        0) -->  Invalid (VPN 0 not valid)
```

此处参数设置不合理，设置的地址空间太小了，可以供存放的页表太小了，存储内容少，甚至无法存储。

./paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2 -c



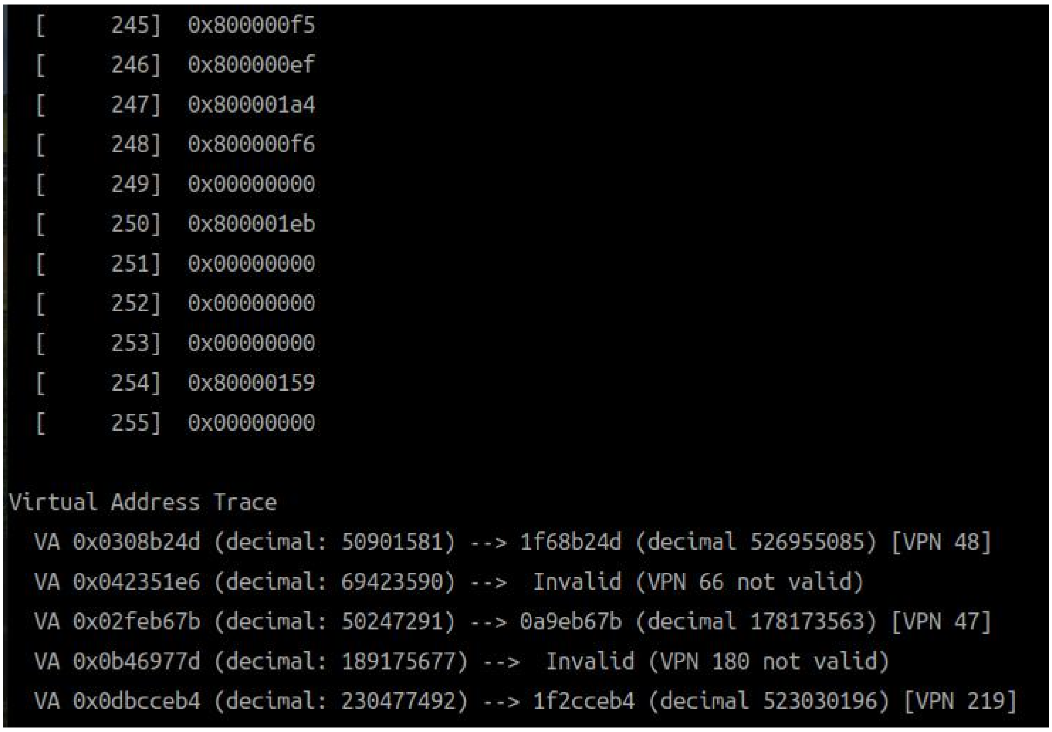
```
zlz@zlz-virtual-machine:~$ ./paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2 -c
ARG seed 2
ARG address space size 32k
ARG phys mem size 1m
ARG page size 8k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]  0x80000079
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x8000005e

Virtual Address Trace
  VA 0x000055b9 (decimal:    21945) --> Invalid (VPN 2 not valid)
  VA 0x00002771 (decimal:    10097) --> Invalid (VPN 1 not valid)
  VA 0x00004d8f (decimal:    19855) --> Invalid (VPN 2 not valid)
  VA 0x00004dab (decimal:    19883) --> Invalid (VPN 2 not valid)
  VA 0x00004a64 (decimal:    19044) --> Invalid (VPN 2 not valid)
```

页面大小为8k，地址空间大小为32k，物理内存大小为1m，页数为4，设置随机种子为2

```
Virtual Address Trace
  VA 0x000055b9 (decimal:    21945) --> Invalid (VPN 2 not valid)
  VA 0x00002771 (decimal:    10097) --> Invalid (VPN 1 not valid)
  VA 0x00004d8f (decimal:    19855) --> Invalid (VPN 2 not valid)
  VA 0x00004dab (decimal:    19883) --> Invalid (VPN 2 not valid)
  VA 0x00004a64 (decimal:    19044) --> Invalid (VPN 2 not valid)
```

第二个情况参数和第一个类似，地址空间相对于页面大小太小了，页数太少了，存储内容少。

./paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3 -c



```
  [     245]  0x800000f5
  [     246]  0x800000ef
  [     247]  0x800001a4
  [     248]  0x800000f6
  [     249]  0x00000000
  [     250]  0x800001eb
  [     251]  0x00000000
  [     252]  0x00000000
  [     253]  0x00000000
  [     254]  0x80000159
  [     255]  0x00000000

Virtual Address Trace
  VA 0x0308b24d (decimal: 50901581) --> 1f68b24d (decimal 526955085) [VPN 48]
  VA 0x042351e6 (decimal: 69423590) -->  Invalid (VPN 66 not valid)
  VA 0x02feb67b (decimal: 50247291) --> 0a9eb67b (decimal 178173563) [VPN 47]
  VA 0x0b46977d (decimal: 189175677) -->  Invalid (VPN 180 not valid)
  VA 0x0dbcceb4 (decimal: 230477492) --> 1f2cceb4 (decimal 523030196) [VPN 219]
```

使用页面大小为1m，地址空间大小为256m，物理内存大小为512m，页数为256，设置随机种子为3

Virtual Address Trace
VA 0x0308b24d(decimal: 50901581) --> 1f68b24d(decimal 526955085)[VPN 48]
VA 0x042351e6(decimal: 69423590) --> Invalid(VPN 66 not valid)
VA 0x02feb67b(decimal: 50247291) --> 0a9eb67b(decimal 178173563)[VPN 47]
VA 0x0b46977d(decimal: 189175677) --> Invalid(VPN 180 not valid)
VA 0x0dbcceb4(decimal: 230477492)-->1f2cceb4 (decimal 523030196)[VPN 219]
一个页的大小为1mb，非常耗费空间的,造成浪费。