18

Q 1

分析1

因为有如下公式成立:

$$page-tablesize = \frac{addressspace}{pagesize}$$

其中`page-table size`是页表大小, `address space`是地址大小, `page size`是页大小

所以知道, 页表大小和地址大小成正比, 与页大小成反比.

因此不可以使用很大的页, 这样会造成很大的浪费.

答案

可以观察题目所给的6条指令的输出变化:

线性页表大小随着地址空间的增长而变化:

- 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

线性页面大小随页大小的增长而变化:

- 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

最终发现输出结果的规律和公式描述的一致

Q 2

分析2

这道题先阅读README查看`-u`:

> 0%（-u 0）到100%（-u 100）。 默认值为50，这意味着虚拟地址空间中大约有1/2页是有效的.

这样的话,肯定是`-u`越大命中几率越大.因此应该是 随着`-u`的设置值的增大而有效地址的个数增加

查看一下指令的`-c`来对比分析结果

```
paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0
paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25
paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50
paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75
paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100
```

答案2

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter18/HW-Paging-LinearTranslate$ python2 ./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)
```

看到`-u 0`的时候是全部非法.

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter18/HW-Paging-LinearTranslate$ python2 ./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]
```

看到`-u 50`的时候,出现了部分合法的.

```
syy@YYshi:/mnt/d/GitHub/lab-junior-sup/lab-junior-sup/os/homework/chapter18/HW-Paging-LinearTranslate$ python2 ./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]
```

看到`-u 100`, 已经全部合法

Q 3

分析3

这道题问的是那些组合是不现实的. 判断现不现实的依据就是第一个问中归纳的公式

$$page - tablesize = \frac{addressspace}{pagesize}$$

移项可得

$$pagesize = \frac{addressspace}{page - tablesize}$$

通过上述公式,计算出下面每个指令的`page size`.

$$shell ./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1 ./paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2 ./paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3 $$

分别得到的结果页表的大小：4、4、256。

但是对应的实际物理内存分别 1024 1m 512m

这么看来，第一个和第三个都不现实。第三个的页表大小太大了，而第一个的页表数不是很多，但是页本身太小了，对应于实际物理地址的1024太小也不合适。