第十八章-分页_介绍

预备知识

在本作业中，您将使用一个简单的程序（paging-linear-translate.py）来检查你是否了解简单的虚拟到物理地址转换如何与线性页表一起使用。 要运行该程序，请输入`python pages-linear-translate.py`。
当使用-h（帮助）标志运行它时，您将看到：

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

Options:
-h, --help              显示帮助信息
-s SEED, --seed=SEED    随机种子
-a ASIZE, --asize=ASIZE
                        地址空间大小(e.g., 16, 64k, ...)
-p PSIZE, --physmem=PSIZE
                        物理空间大小 (e.g., 16, 64k, ...)
-P PAGESIZE, --pagesize=PAGESIZE
                        页的大小(e.g., 4k, 8k, ...)
-n NUM, --addresses=NUM 生成的虚拟地址数量
-u USED, --used=USED    映射的有效百分比
-v                      详细模式
-c                      计算答案
```

首先不使用任何参数运行:

```
>python paging-linear-translate.py
ARG seed 0
ARG address space size 16k
ARG phys mem size 64k
ARG page size 4k
ARG verbose False
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)
   0x8000000c
   0x00000000
   0x00000000
   0x80000006
```

```
Virtual Address Trace
  VA 0x00003229 (decimal:    12841) --> PA or invalid address?
  VA 0x00001369 (decimal:     4969) --> PA or invalid address?
  VA 0x00001e80 (decimal:     7808) --> PA or invalid address?
  VA 0x00002556 (decimal:     9558) --> PA or invalid address?
  VA 0x00003a1e (decimal:    14878) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is an out-of-bounds address (e.g., segfault).
```

如您所见，该程序为您提供的是特定进程的页表（请记住，在具有线性页表的真实系统中，每个进程有一个页表；这里我们仅关注一个进程，其地址空间，因此是一个单页表）。
页表告诉你地址空间的虚拟页号（VPN），虚拟页已映射到的特定的物理帧号（PFN），页号有效或无效。

页表项的格式很简单：最左边的（高阶）位是有效位；其余位（如果有效位为 1）为 PFN。

在上面的示例中，页表将 VPN 0 映射到 PFN 0xc（十进制 12），将 VPN 3 映射到 PFN 0x6（十进制 6），虚拟页 1 和 2 无效(高位为 0)。

如果使用详细标志（-v）运行，会将 VPN（索引）打印到页表中。 在上面的示例中，使用-v 标志运行：

```
Page Table (from entry 0 down to the max size)
  [       0]   0x8000000c
  [       1]   0x00000000
  [       2]   0x00000000
  [       3]   0x80000006
```

然后，你需要做的是使用此页表将打印结果中提供给您的虚拟地址转换为物理地址。让我们看第一个：`VA 0x3229`。 要将这个虚拟地址转换为物理地址，我们首先必须将其分解为虚拟页号和偏移量。我们通过观察地址空间和页面的大小来做到这一点。在此示例中，地址空间设置为 16KB（非常小的地址空间），页面大小为 4KB。因此，我们知道虚拟地址中有 14 位，偏移量为 12 位，为 VPN 留了 2 位。因此，使用我们的地址 0x3229（二进制 11 0010 0010 1001），我们知道前两位指定了 VPN。因此，0x3229 位于虚拟页面 3 上，偏移量为 0x229。

接下来，我们在页表中查看 VPN 3 是否有效, 并映射到某个物理帧，我们可以看到它是有效的（高位为 1）并映射到物理页 6。因此，我们可以通过物理页 6 并将其加上偏移量来获得最终的物理地址，如下所示：
0x6000（物理页，转义到适当的位置）或 0x0229（偏移量），得出最终物理地址：0x6229。因此，在此示例中，我们可以看到虚拟地址 0x3229 转换为物理地址 0x6229。

要查看其余的解决方案，只需使用-c 标志（一如既往）即可：

```
VA  0: 00003229 (decimal: 12841) --> 00006229 (25129) [VPN 3]
VA  1: 00001369 (decimal:  4969) --> Invalid (VPN 1 not valid)
VA  2: 00001e80 (decimal:  7808) --> Invalid (VPN 1 not valid)
VA  3: 00002556 (decimal:  9558) --> Invalid (VPN 2 not valid)
VA  4: 00003a1e (decimal: 14878) --> 00006a1e (27166) [VPN 3]
```

Problem1

问题描述

在做地址转换之前，让我们用模拟器来研究线性页表在给定不同参数的情况下如何改变大小。在不同参数变化时，计算线性页表的大小。一些建议输入如下，通过使用 -v 标志，你可以看到填充了多少个页表项。

首先，要理解线性页表大小如何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
```

然后，理解线性页面大小如何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
```

在运行这些命令之前，请试着想想预期的趋势。页表大小如何随地址空间的增长而改变？随着页大小的增长呢？为什么一般来说，我们不应该使用很大的页呢？

问题分析

页表大小 = 地址空间大小/页面大小*sizeof(PET)

问题解答

1. 页大小为 1k，地址空间大小为 1m
   所以页数为 1m/1k = 1024
2. 页大小为 1k，地址空间大小为 2m
   所以页数为 2m/1k = 2048
3. 页大小为 1k，地址空间大小为 4m
   所以页数为 4m/1k = 4096
4. 页大小为 1k，地址空间大小为 1m
   所以页数为 1m/1k = 1024
5. 页大小为 2k，地址空间大小为 1m
   所以页数为 1m/2k = 512
6. 页大小为 4k，地址空间大小为 1m
   所以页数为 1m/4k = 256

当使用很大的页时会造成大量空间被浪费(页的空间没有被使用完也不能再分割)。

答案验证

```
>python paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
Page Table (from entry 0 down to the max size)
  [       0]   0x8006104a
  ...
  [    1022]   0x00000000
  [    1023]   0x00000000
```

```
>python paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0
Page Table (from entry 0 down to the max size)
  [       0]   0x8006104a
  ...
  [    2046]   0x8000eedd
  [    2047]   0x00000000
```

```
>python paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0
Page Table (from entry 0 down to the max size)
  [       0]   0x8006104a
  ...
  [    4094]   0x00000000
  [    4095]   0x8002e298
```

```
>python paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
Page Table (from entry 0 down to the max size)
  [       0]   0x8006104a
  ...
  [    1022]   0x00000000
  [    1023]   0x00000000
```

```
>python paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
Page Table (from entry 0 down to the max size)
  [       0]   0x80030825
  ...
  [     510]   0x00000000
  [     511]   0x00000000
```

```
>python paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0
Page Table (from entry 0 down to the max size)
  [       0]   0x80018412
  ...
```

```
  [     254]   0x80019c37
  [     255]   0x8001fb27
```

经验证，结果正确。

Problem2

问题描述

现在让我们做一些地址转换。从一些小例子开始，使用 -u 标志更改分配给地址空间的页数。例如：

```
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
```

如果增加每个地址空间中的页的百分比，会发生什么?

问题分析

```
>python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0
ARG seed 0
ARG address space size 16k
ARG phys mem size 32k
ARG page size 1k
ARG verbose True
ARG addresses -1
```

地址空间大小为16k，需要14位，页面大小为1k，需要10位，所以VA高4位为VPN，其余位为offset。物理内存大小为32k，需要15位，所以PA高5位为PFN，其余位为offset。

问题解答

```
>python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0
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) --> PA or invalid address?
  VA 0x00003ee5 (decimal:    16101) --> PA or invalid address?
  VA 0x000033da (decimal:    13274) --> PA or invalid address?
  VA 0x000039bd (decimal:    14781) --> PA or invalid address?
  VA 0x000013d9 (decimal:     5081) --> PA or invalid address?
```

观察Page Table发现页表项有效位（最高位）均为0，所以转换结果均为invalid address.

```
>python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25
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) --> PA or invalid address? VA=0x00003986=11100110000110B
  VPN=1110B=14
  PTE有效位为0
  invalid address
- VA 0x00002bc6 (decimal: 11206) --> PA or invalid address? VA=0x00002bc6=10101111000110B
  VPN=1010B=10
  PTE有效位为1
  offset=1111000110B
  PFV=0x13=10011B
  PA=(PFV<<PFV_shift)|offset=100111111000110B=0x00004fc6=20422
- VA 0x00001e37 (decimal: 7735) --> PA or invalid address? VA=0x00001e37=01111000110111B
  VPN=0111B=7
  PTE有效位为0
  invalid address
- VA 0x00000671 (decimal: 1649) --> PA or invalid address? VA=0x00000671=00011001110001B
  VPN=0001B=1
  PTE有效位为0
  invalid address
- VA 0x00001bc9 (decimal: 7113) --> PA or invalid address? VA=0x00001bc9=01101111001001B
  VPN=0110B=6
  PTE有效位为0
  invalid address

由VA->PA的计算方法相同，为减少重复工作量， 直接使用 -c 参数计算余下情况。

```
>python 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]
```

```
>python 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]
```

```
>python 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]
```

不难发现，随着 -u 参数增加，有效映射的百分比增加，VA->PA的转换成功率随之增加。

答案验证

```
>python 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)
```

```
>python 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)
```

经验证，结果正确。

Problem3

问题描述

现在让我们尝试一些不同的随机种子，以及一些不同的（有时相当疯狂的）地址空间参数：

```
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
```

哪些参数组合是不现实的？为什么？

问题分析

页表大小 = 地址空间大小/页面大小*sizeof(PET)

问题解答

假设sizeof(PET)=4字节

- paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1
  每个进程的页表大小(16)对于物理内存(1024)是友好的，但是地址空间过小，进程可利用空间过少，同时整个地址空间只能被分为4个虚拟页，页数过少导致内存的管理十分不灵活。
- paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2
  每个进程的页表大小(16)对于物理内存(1m)是友好的，但是整个地址空间只能被分为4个虚拟页，页数过少导致内存的管理十分不灵活。
- paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3
  每个进程的页表大小(1024)对于物理内存(512m)是友好的，但是页太大(Linux 页的大小为 4k),导致太多空间被浪费。

综上，上述三个参数组合都是不现实的。