Laboratorio 6

Santiago Osorio Castañeda
Rebeca Servellón Orellana

Homework (Simulation)
In this homework, you will use a simple program, which is known as paging-linear-translate.py, to see if you understand how simple virtual-to-physical address translation works with linear page tables. See the README for details.

1. Before doing any translations, let's use the simulator to study how linear page tables change size given different parameters. Compute the size of linear page tables as different parameters change. Some suggested inputs are below; by using the -v flag, you can see how many page-table entries are filled. First, to understand how linear page table size changes as the address space grows, run with these flags:

-P 1k -a 1m -p 512m -v -n 0

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 1k -a 1m -p 512 -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512
ARG page size 1k
ARG verbose True
ARG addresses -1
```

-P 1k -a 2m -p 512m -v -n 0

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0
ARG seed 0
ARG address space size 2m
ARG phys mem size 512m
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]  0x8006104a
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x80033d4e
  [       4]  0x80026d2f
  [       5]  0x00000000
  [       6]  0x800743d0
  [       7]  0x80024134
  [       8]  0x8004f26b
  [       9]  0x00000000
  [      10]  0x8007dcbe
  [      11]  0x800737a2
  [      12]  0x00000000
  [      13]  0x800730d2
  [      14]  0x8003c6f2
  [      15]  0x00000000
```

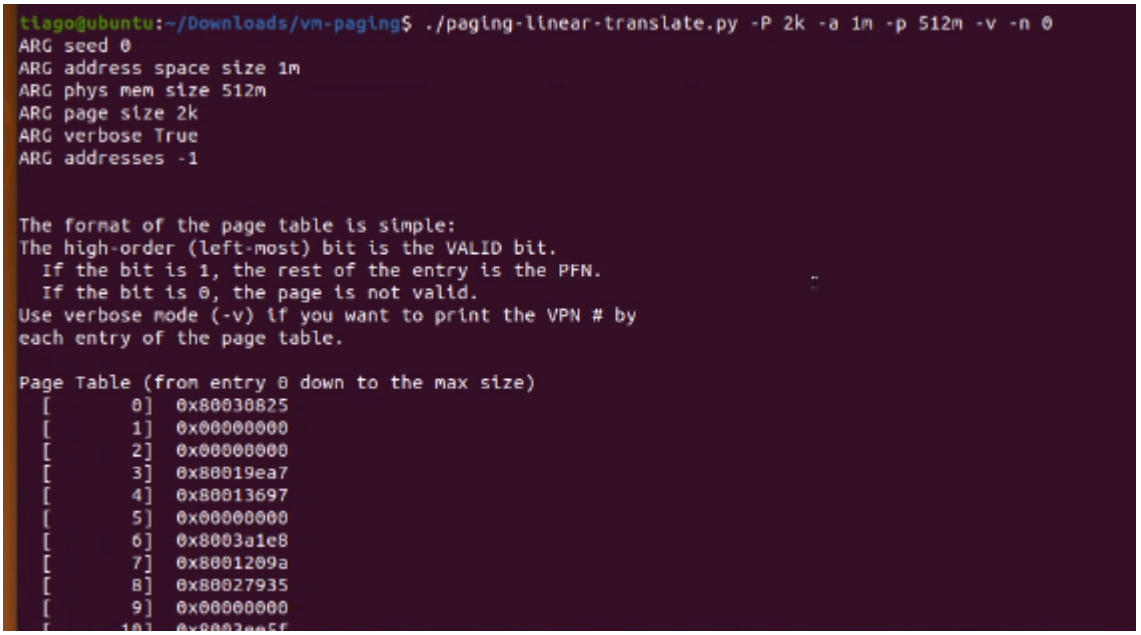
```
  [     2021]  0x00000000
  [     2022]  0x00000000
  [     2023]  0x00000000
  [     2024]  0x00000000
  [     2025]  0x00000000
  [     2026]  0x00000000
  [     2027]  0x8007184d
  [     2028]  0x00000000
  [     2029]  0x8006187f
  [     2030]  0x8001895e
  [     2031]  0x00000000
  [     2032]  0x00000000
  [     2033]  0x00000000
  [     2034]  0x00000000
  [     2035]  0x8002bfac
  [     2036]  0x00000000
  [     2037]  0x8005a39f
  [     2038]  0x8003fa4e
  [     2039]  0x00000000
  [     2040]  0x80038ed5
  [     2041]  0x00000000
  [     2042]  0x00000000
  [     2043]  0x00000000
  [     2044]  0x00000000
  [     2045]  0x00000000
  [     2046]  0x8000eedd
  [     2047]  0x00000000

Virtual Address Trace

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).
```

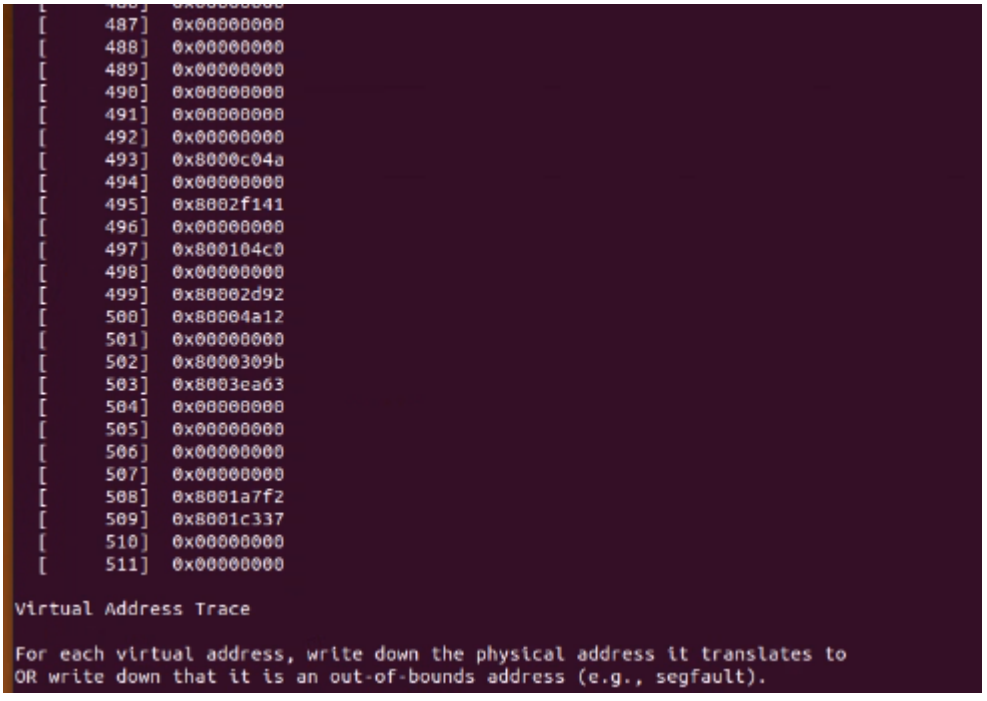
-P 1k -a 4m -p 512m -v -n 0

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0
ARG seed 0
ARG address space size 4m
ARG phys mem size 512m
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]  0x8006104a
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x80033d4e
  [       4]  0x80026d2f
  [       5]  0x00000000
  [       6]  0x800743d0
  [       7]  0x80024134
  [       8]  0x8004f26b
  [       9]  0x00000000
  [      10]  0x8007dcbe
  [      11]  0x800737a2
  [      12]  0x00000000
  [      13]  0x800730d2
```

Then, to understand how linear page table size changes as page size grows:

-P 1k -a 1m -p 512m -v -n 0





-P 2k -a 1m -p 512m -v -n 0

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512m
ARG page size 2k
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]  0x80030825
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x80019ea7
  [       4]  0x80013697
  [       5]  0x00000000
  [       6]  0x8003a1e8
  [       7]  0x8001209a
  [       8]  0x80027935
```

```
  [     483]  0x00000000
  [     484]  0x00000000
  [     485]  0x80032ddd
  [     486]  0x00000000
  [     487]  0x00000000
  [     488]  0x00000000
  [     489]  0x00000000
  [     490]  0x00000000
  [     491]  0x00000000
  [     492]  0x00000000
  [     493]  0x8000c04a
  [     494]  0x00000000
  [     495]  0x8002f141
  [     496]  0x00000000
  [     497]  0x800104c0
  [     498]  0x00000000
  [     499]  0x80002d92
  [     500]  0x80004a12
  [     501]  0x00000000
  [     502]  0x8000309b
  [     503]  0x8003ea63
  [     504]  0x00000000
  [     505]  0x00000000
  [     506]  0x00000000
  [     507]  0x00000000
  [     508]  0x8001a7f2
  [     509]  0x8001c337
  [     510]  0x00000000
  [     511]  0x00000000

Virtual Address Trace
```

-P 4k -a 1m -p 512m -v -n 0

```
tlago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0
ARG seed 0
ARG address space size 1m
ARG phys mem size 512m
ARG page size 4k
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]  0x80018412
  [       1]  0x00000000
  [       2]  0x00000000
  [       3]  0x8000cf53
  [       4]  0x80009b4b
  [       5]  0x00000000
  [       6]  0x8001d0f4
  [       7]  0x8000904d
  [       8]  0x80013c9a
  [       9]  0x00000000
  [      10]  0x8001f72f
  [      11]  0x8001cde8
  [      12]  0x00000000
  [      13]  0x8001cc34
  [      14]  0x8000f1bc
  [      15]  0x00000000
  [      16]  0x00000000
  [      17]  0x8001d376
```

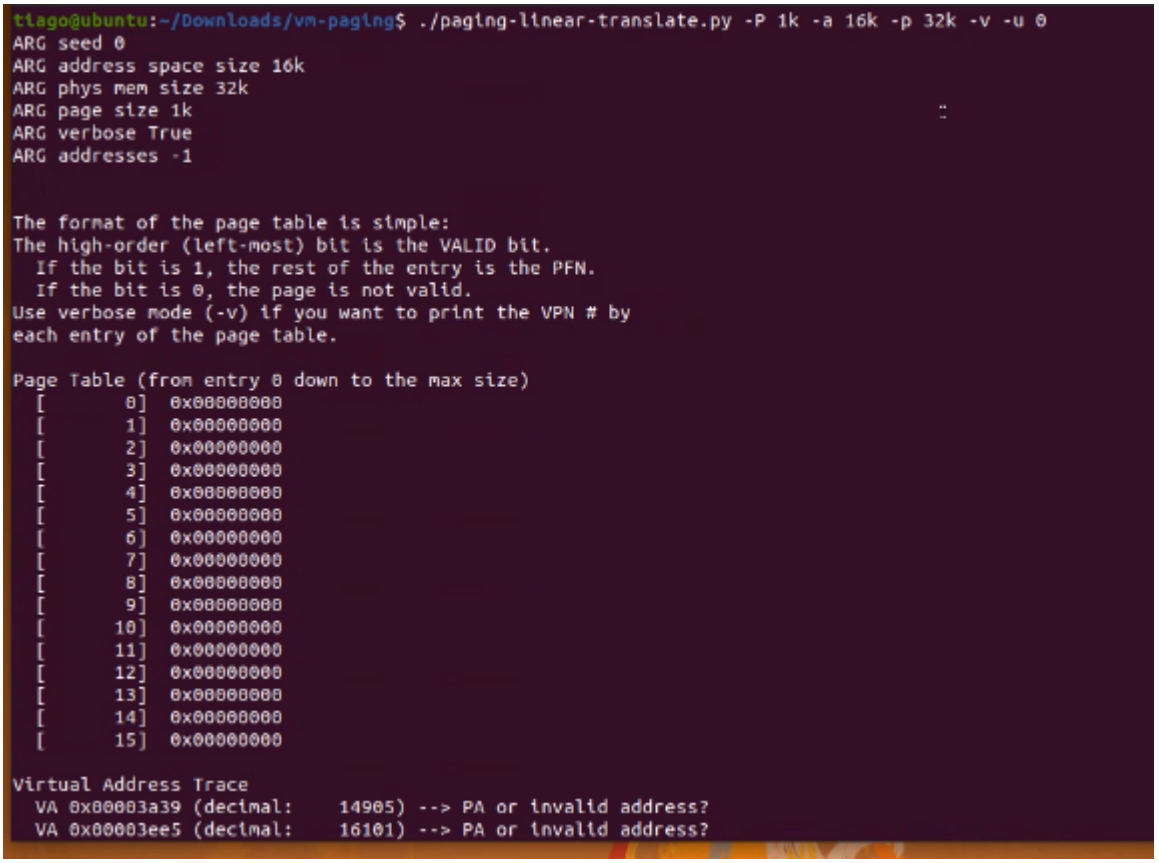
```
  [     229]  0x00000000
  [     230]  0x8000a519
  [     231]  0x00000000
  [     232]  0x00000000
  [     233]  0x8000676b
  [     234]  0x80007003
  [     235]  0x00000000
  [     236]  0x8001cc4a
  [     237]  0x8000122b
  [     238]  0x00000000
  [     239]  0x00000000
  [     240]  0x8001af46
  [     241]  0x80016fae
  [     242]  0x800021f9
  [     243]  0x00000000
  [     244]  0x8000142e
  [     245]  0x00000000
  [     246]  0x00000000
  [     247]  0x00000000
  [     248]  0x8000a943
  [     249]  0x00000000
  [     250]  0x00000000
  [     251]  0x8001efec
  [     252]  0x8001cd5b
  [     253]  0x800125d2
  [     254]  0x80019c37
  [     255]  0x8001fb27

Virtual Address Trace

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).
```

Before running any of these, try to think about the expected trends. How should page-table size change as the address space grows? As the page size grows? Why not use big pages in general?

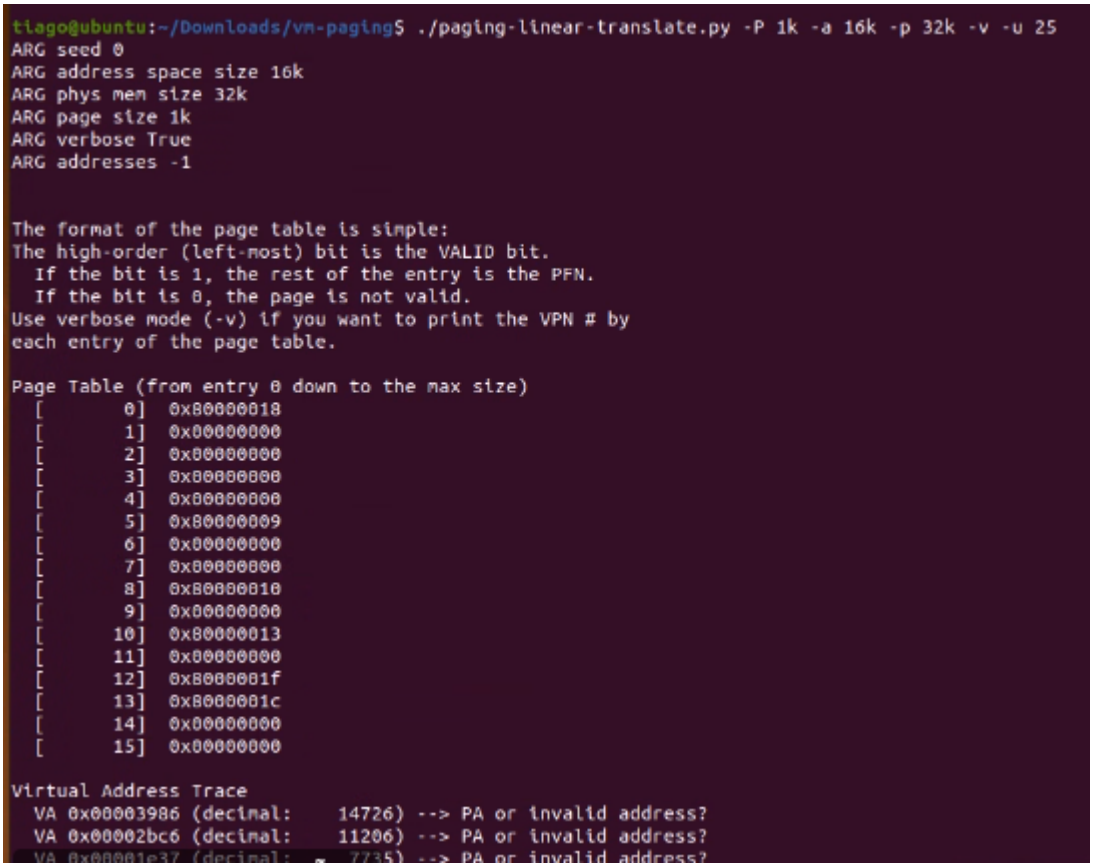
Cuando los tamaños de página aumentan, el tamaño de la tabla de páginas disminuye porque necesitamos menos páginas (porque son más grandes en tamaño) para cubrir todo el espacio de direcciones.
No se debe establecer tamaños de página muy grandes para evitar el desperdicio de espacio en la memoria que puede terminar siendo un problema de fragmentación interna en caso que el proceso no la llegue a utilizar toda.

2. Now let's do some translations. Start with some small examples, and change the number of pages that are allocated to the address space with the -u flag. For example:

-P 1k -a 16k -p 32k -v -u 0



-P 1k -a 16k -p 32k -v -u 25



-P 1k -a 16k -p 32k -v -u 50

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50
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) --> PA or invalid address?
  VA 0x0000231d (decimal:     8989) --> PA or invalid address?
  VA 0x000008e6 (decimal:     2278) --> PA or invalid address?
```

-P 1k -a 16k -p 32k -v -u 75

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75
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) --> PA or invalid address?
  VA 0x00001986 (decimal:     6534) --> PA or invalid address?
```

-P 1k -a 16k -p 32k -v -u 100

```
tlago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100
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) --> PA or invalid address?
  VA 0x00001986 (decimal:     6534) --> PA or invalid address?
  VA 0x000034ca (decimal:    13514) --> PA or invalid address?
```

What happens as you increase the percentage of pages that are allocated in each address space?
Las direcciones de memoria que cubren cada página se vuelven válidas

3. Now let’s try some different random seeds, and some different (and sometimes quite crazy) address-space parameters, for variety:

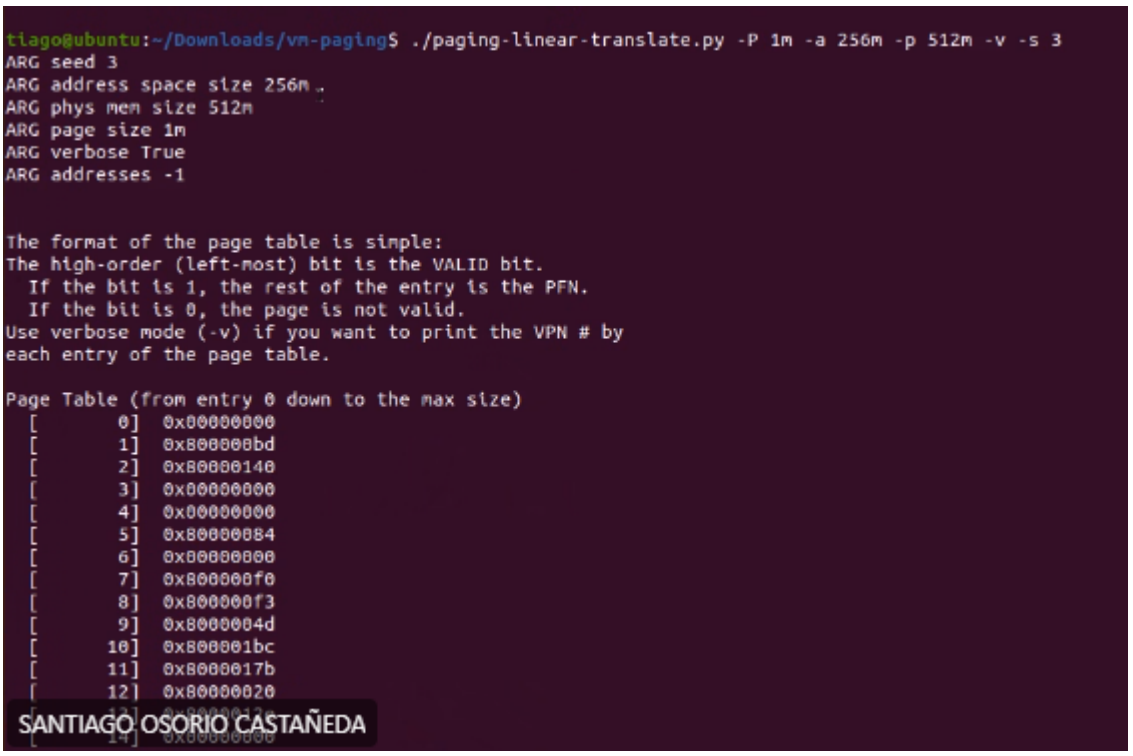
-P 8 -a 32 -p 1024 -v -s 1

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1
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) --> PA or invalid address?
  VA 0x00000014 (decimal:       20) --> PA or invalid address?
  VA 0x00000019 (decimal:       25) --> PA or invalid address?
  VA 0x00000003 (decimal:        3) --> PA or invalid address?
  VA 0x00000000 (decimal:        0) --> 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).
```

-P 8k -a 32k -p 1m -v -s 2

```
tiago@ubuntu:~/Downloads/vm-paging$ ./paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2
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) --> PA or invalid address?
  VA 0x00002771 (decimal:    10097) --> PA or invalid address?
  VA 0x00004d8f (decimal:    19855) --> PA or invalid address?
  VA 0x00004dab (decimal:    19883) --> PA or invalid address?
  VA 0x00004a64 (decimal:    19044) --> 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).
```

-P 1m -a 256m -p 512m -v -s 3



Which of these parameter combinations are unrealistic? Why?
Los de tamaño más pequeño

4. Use the program to try out some other problems. Can you find the limits of where the program doesn't work anymore? For example, what happens if the address-space size is bigger than physical memory?

- No funciona porque la memoria física debe tener la capacidad de administrar las direcciones del address space.

```
Error: physical memory size must be GREATER than address space size (for this simulation)
```

- Generalmente el tamaño de las páginas debe ser igual al del frame

- Cuando la memoria física no es múltiplo del tamaño de la página.

- Cuando el address space no es múltiplo del tamaño de la página.

```
Error in argument: address space must be a multiple of the pagesize
tiago@ubuntu:~/Downloads/vm-paging$
```

- Espacio de memoria con valores negativos

```
Error: must specify a non-zero address-space size.
```