Tobias Sautter BSYS HW5 **22.04.2023**

Question 1

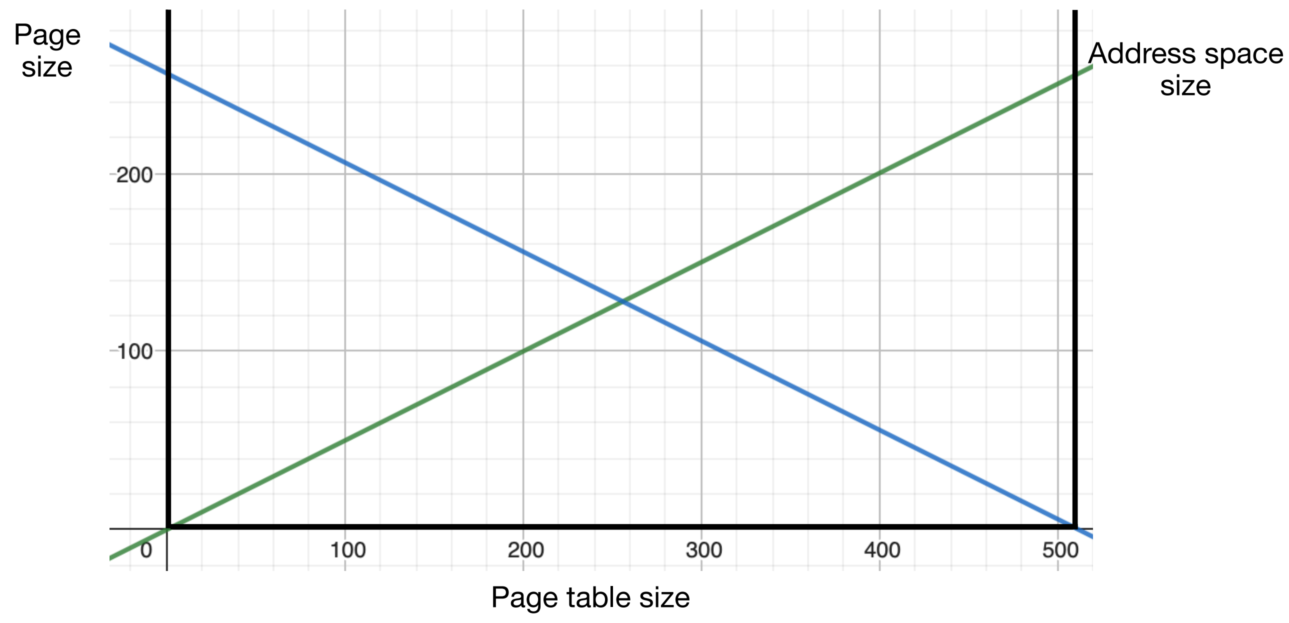
Before doing any translations, let's use the simulator to study how linear page tables change size given different parameters. Compute the size of linear page tables as different parameters change. Some suggested inputs are below; by using the -v flag, you can see how many page-table entries are filled.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

```
python3 paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
python3 paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0
python3 paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0
```

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

```
python3 paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
python3 paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
python3 paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0
```

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?



Question 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:

```
python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0
python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25
python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50
python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75
python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100
```

What happens as you increase the percentage of pages that are allocated in each address space?

python3 paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0

Every address is invalid since no page is allocated to the address space.

```
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


Offset:
16KB = 16384 => log2(16384) = 14
1 KB = 1024  => log2(1024) = 10

Page table bytes:
14 - 10 = 4


Virtual Address Trace
  VA 0x00003986 (decimal:    14726) --> 1110 0110000110
     1110 0110000110 => VPN: 14 Offset: 390
     VPN 14 is invalid

  VA 0x00002bc6 (decimal:    11206) --> 1010 1111000110
     1010 1111000110 => VPN: 10 Offset: 966
     VPN: 10 => 0x13 <<10 = 0100110000000000
     0100110000000000(0x13) OR 1111000110(Offset) = 0100111111000110 = 20422

  VA 0x00001e37 (decimal:     7735) --> 0111 1000110111  not valid
  VA 0x00000671 (decimal:     1649) --> 0001 1001110001  not valid
  VA 0x00001bc9 (decimal:     7113) --> 0110 1111001001  not valid
```

With increased percentage, more mem accesses become valid, but the free space decreases.

Question 3

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

```
python3 paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1
python3 paging-linear-translate.py -P 8 k-a 32k -p 1m -v -s 2
python3 paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3
```

Which of these parameter combinations are unrealistic? Why?

```
ARG seed 1
ARG address space size 32
ARG phys mem size 1024
ARG page size 8
ARG verbose True
ARG addresses -1

[       0]  0x00000000
[       1]  0x80000061
[       2]  0x00000000
[       3]  0x00000000
```

```
ARG seed 2
ARG address space size 16k
ARG phys mem size 1m
ARG page size 8
ARG verbose True
ARG addresses -1

[       0]  0x8001e549
[       1]  0x00000000
[    . . . ]
[    2046]  0x00000000
[    2047]  0x8001be52
```

```
ARG seed 3
ARG address space size 256m
ARG phys mem size 512m
ARG page size 1m
ARG verbose True
ARG addresses -1

[       0]  0x00000000
[       1]  0x800000bd
[    . . . ]
[     254]  0x80000159
[     255]  0x00000000
```

The first one is unrealistically small.
The second one has a really small page size for the size of the address space size.
The third one seems fine but the big page size might result in a lot of internal fragmentation.

Question 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?

Page || address space size == 0
Physical mem size <= address space size.
Page size > address space size.