Operating Systems - CS 304

Rishit Saiya - 180010027, Assignment - 3

April 9, 2021

1 Abstract

In this assignment, the main task was to understand and analyse the translation functioning from virtual address to physical address. In order to analyse the affect of different parameters on translation process, 2 different translation techniques were used namely: **Linear Page Table Translation** & **Multilevel Page Table Translation**.

2 Linear Page Table

2.1 Q1

From the lectures and references, the Page Table Size is defined as follows:

$$PT\_Size = \frac{VAS}{P\_size} \times PTE\_size \tag{1}$$

where PT_size is Page Table Size, VAS is Virtual Address Size, P_size is Page Size & PTE_size is Page Table Entry Size.

In the 1^{st} part of the question, analysis on Page Table Size is required when Virtual Address Size is increased. The results pertaining to that are as follows:

Flags added to command	Page Table Size
`-P 1k -a 1m -p 512m -v -n 0`	1024 [0 - 1023]
`-P 1k -a 2m -p 512m -v -n 0`	2048 [0 - 2047]
`-P 1k -a 4m -p 512m -v -n 0`	4096 [0 - 4095]

Table 1: Variation of Page Table Size with Virtual Address Size

The above results (Table 1) were obtained after running the command:
`python linear-page-translate.py <Flags added to command>`. The flags and their functionalities can be found here. From Equation (1), it is clear that Page Table Size is increasing as Virtual Address Space Size increases (PT_size $\propto$ VAS). The following graph (Figure 1) also emphasises on the same point mentioned above:

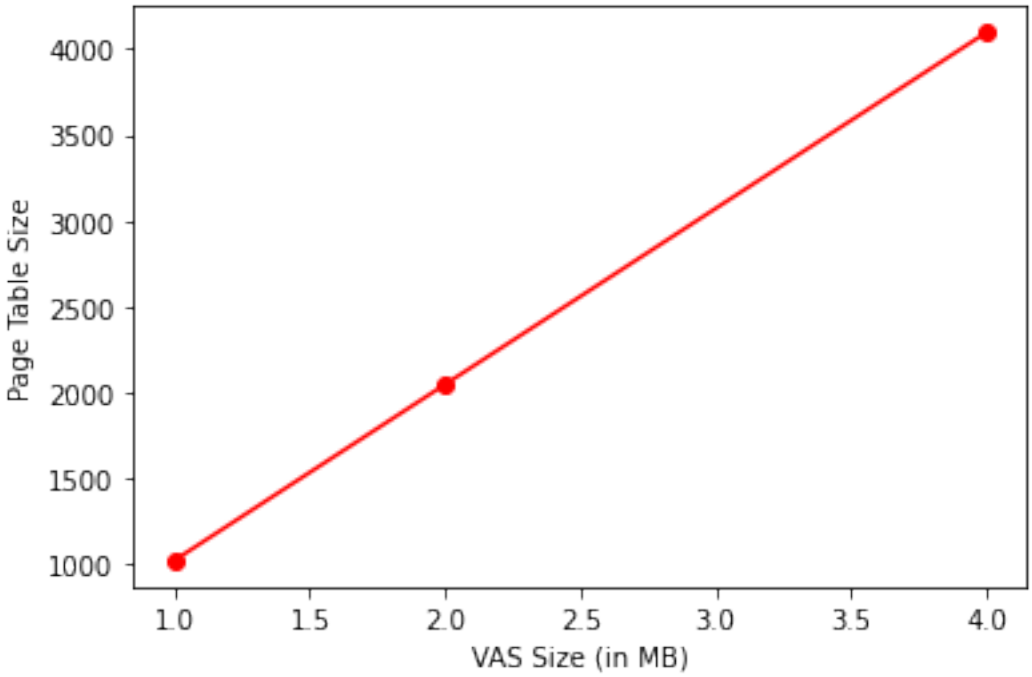

Figure 1: Variation of Page Table Size with Virtual Address Size

In the 2^{nd} part of the question, analysis on Page Table Size is required when Page Size is increased. The results pertaining to that are as follows:

Flags added to command	**Page Table Size**
`-P 1k -a 1m -p 512m -v -n 0`	1024 [0 - 1023]
`-P 2k -a 1m -p 512m -v -n 0`	512 [0 - 511]
`-P 4k -a 1m -p 512m -v -n 0`	256 [0 - 255]

Table 2: Variation of Page Table Size with Page Table Size

The above results (Table 2) were obtained after running the command:
`python linear-page-translate.py <Flags added to command>`. The flags and their functionalities can be found here. It is clear that Page Table Size is decreasing as Page Size increases ($\text{PT\_size} \propto \frac{1}{VAS}$).

The conclusion would be to avoid using pages of large size in general because they cause Internal Fragmentation in the pages. Softwares & Applications end up allocating pages but only using small segments and pieces of each, and memory quickly fills up with these overly-large pages. The following graph (Figure 2) also emphasises on the same point mentioned above:

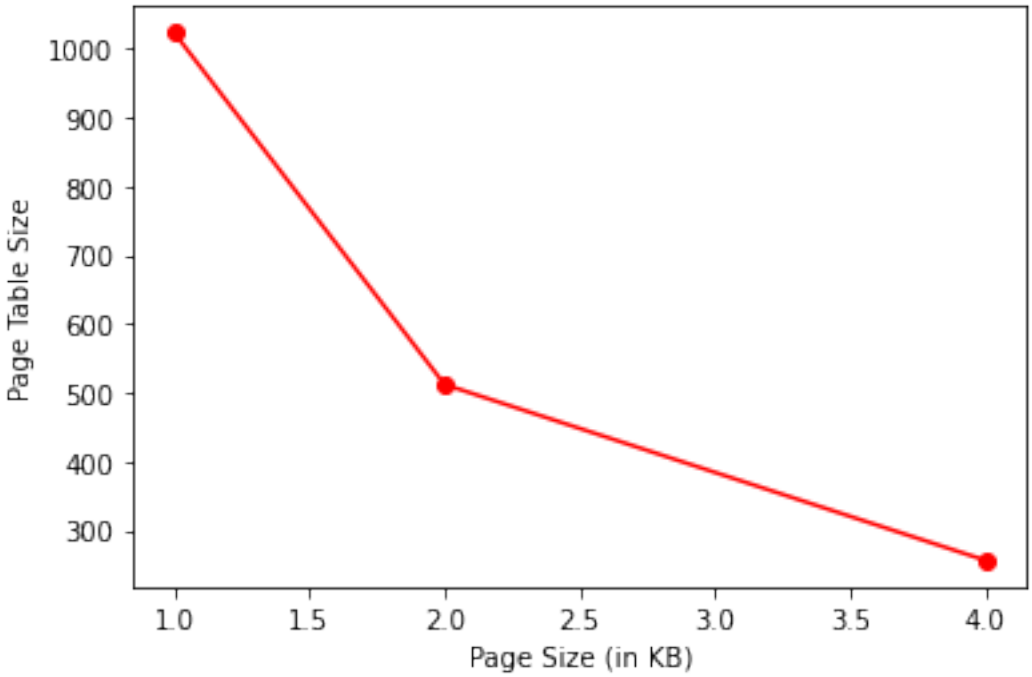

Figure 2: Variation of Page Table Size with Page Table Size

2.2 Q2

In this question, the affect of increase of percentage of pages that are allocated in each address space had to be analysed.

With this reference, `-u` flag represents percentage of the address space used. The domain D of `-u` can be defined as D $\in$ [0, 100]. The extremes of domain can be explained as follows:

- 0 indicates that no address space allocated to the process is in use.
- 100 indicates that full Virtual Address Space is allocated to the process is used.

0% of used address space leads to all entries invalid in VA to PA translation. Thus 0 is illegal to have. As percentage of used address space is increased in terms of %, more and more VA to PA translations become valid in table. Thus, we see as percentage of Address space used increases, more valid entries appear in the table and as it reaches 100%, all entries are valid.

The set of commands used are as follows:

```
-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
-P 1k -a 16k -p 32k -v -u 75
-P 1k -a 16k -p 32k -v -u 100
```

The corresponding outputs are as follows in Figures 3-7.

```
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)
```

Figure 3: Address Space Consumption = 0% $\implies$ `-u 0`

2.3 Q3

In this question, different parameters and some crazy combinations were tried. The results pertaining to that are as follows:

From Equation (1), the following expression can be easily derived:

$$P\_size = \frac{VAS}{PT\_Size} \times PTE\_size$$

With the above data, it is clear that 1^{st} and 3^{rd} are unrealistic in nature. The size of the 3^{rd} page table is too large and correspondingly the number of the 1^{st} page table is not very large. But, also the page size has to be considered to analyse. The page size is small which is actually 1024 units corresponding to the actual physical address and hence is too small to be appropriate in implementation.

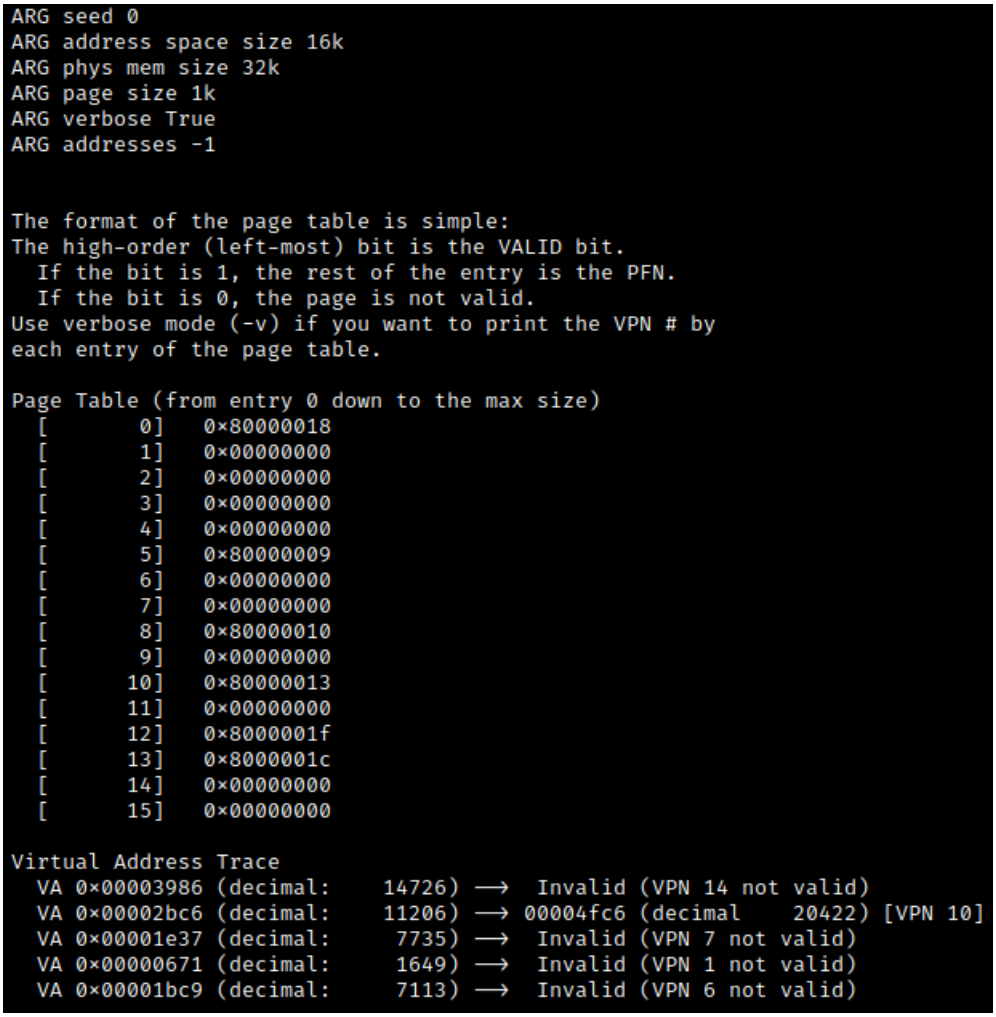

Figure 4: Address Space Consumption = 25% $\implies$ `-u 25`

2.4 Q4

In this question, some boundary conditions had to be found out where the program could possibly show error. So, the following are some of the various boundary conditions that show error while in implementation. (Figures 8-12)

- Considering the Page Size as 0 (Figure 8) and correspondingly using command:
  `python paging-linear-translate.py -P 0 -v -c`
- Considering the Address Space as 0 (Figure 9) and correspondingly using command:
  `python paging-linear-translate.py -a 0 -v -c`
- Considering the Physical Address as 0 (Figure 10) and correspondingly using command:
  `python paging-linear-translate.py -p 0 -v -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]
```

Figure 5: Address Space Consumption = 50% $\implies$ `-u 50`

- Considering the case where Page Size is taken too large, so that it gives error in indexing Virtual Address in Page Table as only one entry (Figure 11) and correspondingly using command:

  `python paging-linear-translate.py -P 32k -v -c`

- Considering the case where Physical Memory Size less than Virtual Address Space Size (Figure 12) and correspondingly using command:

  `python paging-linear-translate.py -a 65k -v -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]
```

Figure 6: Address Space Consumption = 75% $\implies$ `-u 75`

3 Multilevel Page Table

3.1 Q1

In a Linear Page Table, only a single register is required to locate the page table. However this is done with the assumption that hardware does the lookup upon a Translation Lookaside Buffer miss (cache miss). With the similar idea, 2 registers are required to locate a 2 level page table and 3 registers are required to locate a 3 level table.

3.2 Q2

In this question, the program is run for 3 different seeds:

- Case: `Seed = 1` - According to output, there are 10 translations. 5 Translations to

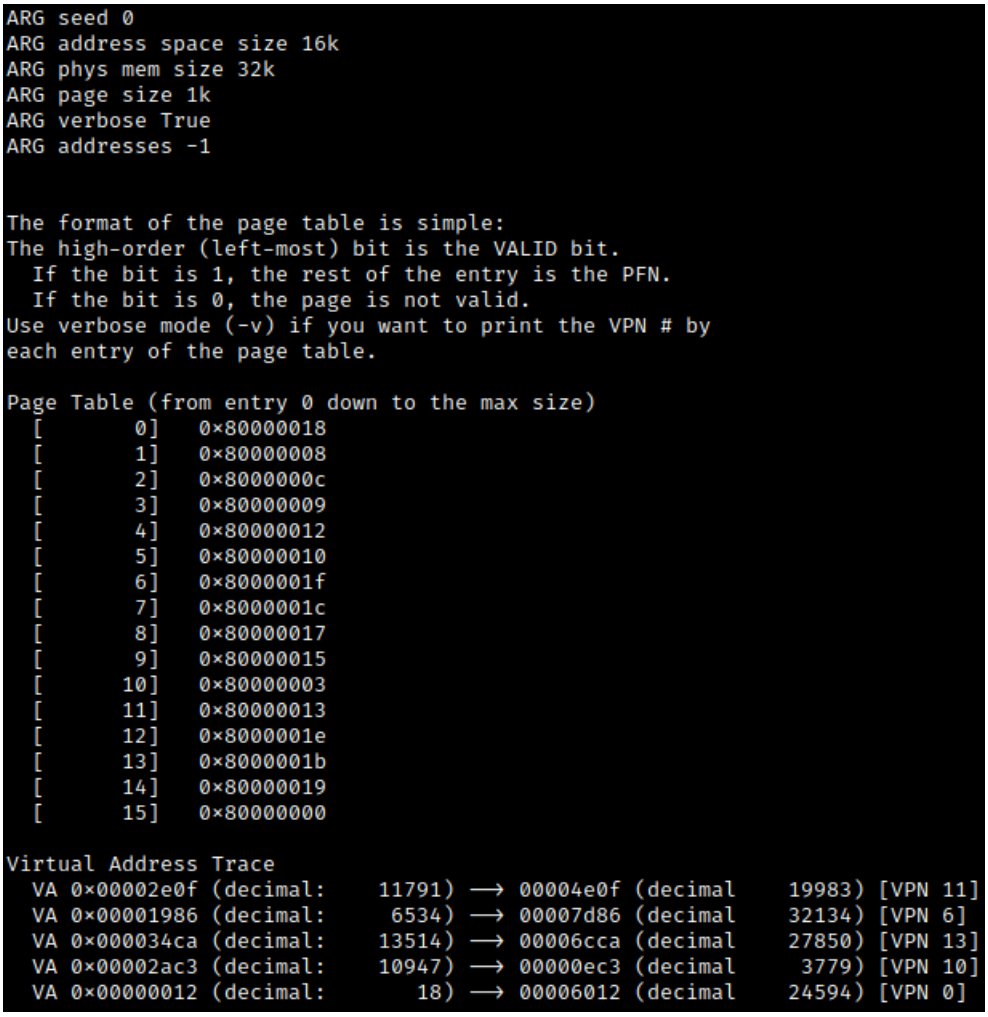

Figure 7: Address Space Consumption = 100% $\implies$ `-u 100`

Physical Addresses, 4 Page Fault (Page table entry not valid) and 1 Page Fault (Page Directory entry not valid). Each Virtual Address to Physical Address translation at most takes 3 Memory Lookups from disks. (Figure 13)

- Case: `Seed = 2` - According to output, there are 10 translations. 5 Translations to Physical Addresses, 3 Page Fault (Page table entry not valid) and 2 Page Fault (Page Directory entry not valid). Each Virtual Address to Physical Address translation at most takes 3 Memory Lookups from disks. (Figure 14)

- Case: `Seed = 3` - According to output, there are 10 translations. 8 Translations to Physical Addresses, 1 Page Fault (Page table entry not valid) and 1 Page Fault (Page Directory entry not valid). Each Virtual Address to Physical Address translation at most takes 3 Memory Lookups from disks. (Figure 15)

Flags added to command	Page Size	Physical Memory
`-P 8 -a 32 -p 1024 -v -s 1`	4	1024
`-P 8k -a 32k -p 1m -v -s 2`	4	1m
`-P 1m -a 256m -p 512m -v -s 3`	256	512m

Table 3: Variation of Parameters randomly

```
└─$ python paging-linear-translate.py -P 0 -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 64k
ARG page size 0
ARG verbose True
ARG addresses -1

Traceback (most recent call last):
  File "paging-linear-translate.py", line 85, in <module>
    mustbemultipleof(asize, pagesize, 'address space must be a multiple of the pagesize')
  File "paging-linear-translate.py", line 14, in mustbemultipleof
    if (int(float(bignum)/float(num)) != (int(bignum) / int(num))):
ZeroDivisionError: float division by zero
```

Figure 8: Error: Page Size considered as 0

3.3 Q3

In this question, the analysis has to be done how cache memory works and also on how memory references in page table behaviour in cache. Since the addresses in above tables are random in nature, the miss rate is relatively higher as mentioned in slides. There will be slow accesses because of unfit temporal and spatial localities. Since the Translation Lookaside Buffer is established, the existence of Temporal and Spatial Localities can boost a higher Hit Rate.

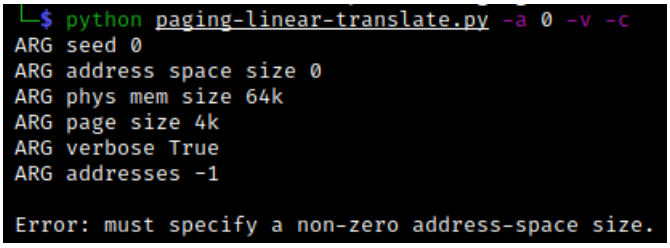

Figure 9: Error: Address Space considered as 0

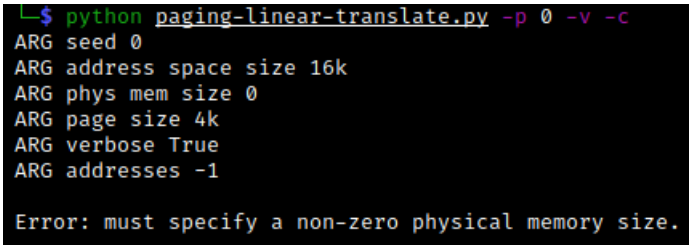

Figure 10: Error: Physical Address considered as 0

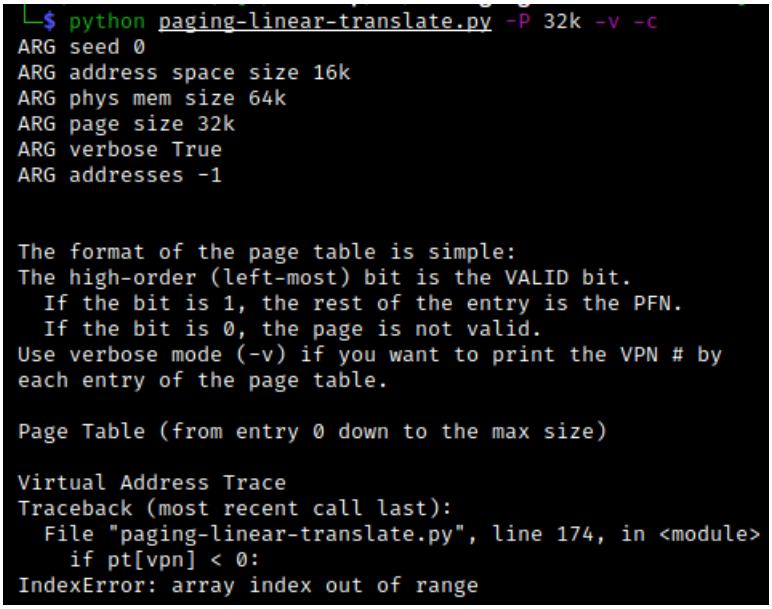

Figure 11: Error: Indexing Misconfiguration executed when Page Size is taken too large

```
└─$ python paging-linear-translate.py -a 65k -v -c
ARG seed 0
ARG address space size 65k
ARG phys mem size 64k
ARG page size 4k
ARG verbose True
ARG addresses -1

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

Figure 12: Error: Mapping Error when Physical Memory Size is taken lesser than Virtual Address Space Size

```
PDBR: 17  (decimal) [This means the page directory is held in this page]

Virtual Address 6c74:
  --> pde index:0x1b [decimal 27] pde contents:0xa0 (valid 1, pfn 0x20 [decimal 32])
    --> pte index:0x3 [decimal 3] pte contents:0xe1 (valid 1, pfn 0x61 [decimal 97])
      --> Translates to Physical Address 0xc34 --> Value: 06
Virtual Address 6b22:
  --> pde index:0x1a [decimal 26] pde contents:0xd2 (valid 1, pfn 0x52 [decimal 82])
    --> pte index:0x19 [decimal 25] pte contents:0xc7 (valid 1, pfn 0x47 [decimal 71])
      --> Translates to Physical Address 0x8e2 --> Value: 1a
Virtual Address 03df:
  --> pde index:0x0 [decimal 0] pde contents:0xda (valid 1, pfn 0x5a [decimal 90])
    --> pte index:0x1e [decimal 30] pte contents:0x85 (valid 1, pfn 0x05 [decimal 5])
      --> Translates to Physical Address 0x0bf --> Value: 0f
Virtual Address 69dc:
  --> pde index:0x1a [decimal 26] pde contents:0xd2 (valid 1, pfn 0x52 [decimal 82])
    --> pte index:0xe [decimal 14] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 317a:
  --> pde index:0xc [decimal 12] pde contents:0x98 (valid 1, pfn 0x18 [decimal 24])
    --> pte index:0xb [decimal 11] pte contents:0xb5 (valid 1, pfn 0x35 [decimal 53])
      --> Translates to Physical Address 0x6ba --> Value: 1e
Virtual Address 4546:
  --> pde index:0x11 [decimal 17] pde contents:0xa1 (valid 1, pfn 0x21 [decimal 33])
    --> pte index:0xa [decimal 10] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 2c03:
  --> pde index:0xb [decimal 11] pde contents:0xc4 (valid 1, pfn 0x44 [decimal 68])
    --> pte index:0x0 [decimal 0] pte contents:0xd7 (valid 1, pfn 0x57 [decimal 87])
      --> Translates to Physical Address 0xae3 --> Value: 16
Virtual Address 7fd7:
  --> pde index:0x1f [decimal 31] pde contents:0x92 (valid 1, pfn 0x12 [decimal 18])
    --> pte index:0x1e [decimal 30] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
Virtual Address 390e:
  --> pde index:0xe [decimal 14] pde contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page directory entry not valid)
Virtual Address 748b:
  --> pde index:0x1d [decimal 29] pde contents:0x80 (valid 1, pfn 0x00 [decimal 0])
    --> pte index:0x4 [decimal 4] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
```

Figure 13: Seed = 1

```
PDBR: 122  (decimal) [This means the page directory is held in this page]

Virtual Address 7570:
  ⟶ pde index:0×1d [decimal 29] pde contents:0×b3 (valid 1, pfn 0×33 [decimal 51])
    ⟶ pte index:0×b [decimal 11] pte contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page table entry not valid)
Virtual Address 7268:
  ⟶ pde index:0×1c [decimal 28] pde contents:0×de (valid 1, pfn 0×5e [decimal 94])
    ⟶ pte index:0×13 [decimal 19] pte contents:0×e5 (valid 1, pfn 0×65 [decimal 101])
      ⟶ Translates to Physical Address 0×ca8 ⟶ Value: 16
Virtual Address 1f9f:
  ⟶ pde index:0×7 [decimal 7] pde contents:0×af (valid 1, pfn 0×2f [decimal 47])
    ⟶ pte index:0×1c [decimal 28] pte contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page table entry not valid)
Virtual Address 0325:
  ⟶ pde index:0×0 [decimal 0] pde contents:0×82 (valid 1, pfn 0×02 [decimal 2])
    ⟶ pte index:0×19 [decimal 25] pte contents:0×dd (valid 1, pfn 0×5d [decimal 93])
      ⟶ Translates to Physical Address 0×ba5 ⟶ Value: 0b
Virtual Address 64c4:
  ⟶ pde index:0×19 [decimal 25] pde contents:0×b8 (valid 1, pfn 0×38 [decimal 56])
    ⟶ pte index:0×6 [decimal 6] pte contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page table entry not valid)
Virtual Address 0cdf:
  ⟶ pde index:0×3 [decimal 3] pde contents:0×9d (valid 1, pfn 0×1d [decimal 29])
    ⟶ pte index:0×6 [decimal 6] pte contents:0×97 (valid 1, pfn 0×17 [decimal 23])
      ⟶ Translates to Physical Address 0×2ff ⟶ Value: 00
Virtual Address 2906:
  ⟶ pde index:0×a [decimal 10] pde contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page directory entry not valid)
Virtual Address 7a36:
  ⟶ pde index:0×1e [decimal 30] pde contents:0×8a (valid 1, pfn 0×0a [decimal 10])
    ⟶ pte index:0×11 [decimal 17] pte contents:0×e6 (valid 1, pfn 0×66 [decimal 102])
      ⟶ Translates to Physical Address 0×cd6 ⟶ Value: 09
Virtual Address 21e1:
  ⟶ pde index:0×8 [decimal 8] pde contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page directory entry not valid)
Virtual Address 5149:
  ⟶ pde index:0×14 [decimal 20] pde contents:0×bb (valid 1, pfn 0×3b [decimal 59])
    ⟶ pte index:0×a [decimal 10] pte contents:0×81 (valid 1, pfn 0×01 [decimal 1])
      ⟶ Translates to Physical Address 0×029 ⟶ Value: 1b
```

Figure 14: Seed = 2

```
PDBR: 30  (decimal) [This means the page directory is held in this page]

Virtual Address 45b0:
  ⟶ pde index:0×11 [decimal 17] pde contents:0×cd (valid 1, pfn 0×4d [decimal 77])
    ⟶ pte index:0×d [decimal 13] pte contents:0×af (valid 1, pfn 0×2f [decimal 47])
      ⟶ Translates to Physical Address 0×5f0 ⟶ Value: 14
Virtual Address 7075:
  ⟶ pde index:0×1c [decimal 28] pde contents:0×91 (valid 1, pfn 0×11 [decimal 17])
    ⟶ pte index:0×3 [decimal 3] pte contents:0×8c (valid 1, pfn 0×0c [decimal 12])
      ⟶ Translates to Physical Address 0×195 ⟶ Value: 16
Virtual Address 135c:
  ⟶ pde index:0×4 [decimal 4] pde contents:0×e2 (valid 1, pfn 0×62 [decimal 98])
    ⟶ pte index:0×1a [decimal 26] pte contents:0×a5 (valid 1, pfn 0×25 [decimal 37])
      ⟶ Translates to Physical Address 0×4bc ⟶ Value: 11
Virtual Address 3727:
  ⟶ pde index:0×d [decimal 13] pde contents:0×a6 (valid 1, pfn 0×26 [decimal 38])
    ⟶ pte index:0×19 [decimal 25] pte contents:0×dc (valid 1, pfn 0×5c [decimal 92])
      ⟶ Translates to Physical Address 0×b87 ⟶ Value: 13
Virtual Address 48c4:
  ⟶ pde index:0×12 [decimal 18] pde contents:0×a7 (valid 1, pfn 0×27 [decimal 39])
    ⟶ pte index:0×6 [decimal 6] pte contents:0×de (valid 1, pfn 0×5e [decimal 94])
      ⟶ Translates to Physical Address 0×bc4 ⟶ Value: 0d
Virtual Address 22bc:
  ⟶ pde index:0×8 [decimal 8] pde contents:0×b5 (valid 1, pfn 0×35 [decimal 53])
    ⟶ pte index:0×15 [decimal 21] pte contents:0×ac (valid 1, pfn 0×2c [decimal 44])
      ⟶ Translates to Physical Address 0×59c ⟶ Value: 1b
Virtual Address 15ee:
  ⟶ pde index:0×5 [decimal 5] pde contents:0×c3 (valid 1, pfn 0×43 [decimal 67])
    ⟶ pte index:0×f [decimal 15] pte contents:0×d4 (valid 1, pfn 0×54 [decimal 84])
      ⟶ Translates to Physical Address 0×a8e ⟶ Value: 1c
Virtual Address 7bb9:
  ⟶ pde index:0×1e [decimal 30] pde contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page directory entry not valid)
Virtual Address 1ebe:
  ⟶ pde index:0×7 [decimal 7] pde contents:0×e8 (valid 1, pfn 0×68 [decimal 104])
    ⟶ pte index:0×15 [decimal 21] pte contents:0×7f (valid 0, pfn 0×7f [decimal 127])
      ⟶ Fault (page table entry not valid)
Virtual Address 4a10:
  ⟶ pde index:0×12 [decimal 18] pde contents:0×a7 (valid 1, pfn 0×27 [decimal 39])
    ⟶ pte index:0×10 [decimal 16] pte contents:0×e6 (valid 1, pfn 0×66 [decimal 102])
      ⟶ Translates to Physical Address 0×cd0 ⟶ Value: 0c
```

Figure 15: Seed = 3