Operating Systems Laboratory

Lab 7

Table of Contents

Question 1

Part 1

```
./relocation.py -s 1
```

```
Seed  : 1
Base  : 0x0000363c (decimal 13884)
Limit : 290
```

Virtual Address	In Bounds	Translation, if yes
0: 0x0000030e (decimal: 782)	VIOLATION	-
1: 0x00000105 (decimal: 261)	VALID	0x00003741 (decimal: 14145)
2: 0x000001fb (decimal: 507)	VIOLATION	-
3: 0x000001cc (decimal: 460)	VIOLATION	-
4: 0x0000029b (decimal: 667)	VIOLATION	-

```
./relocation.py -s 2
```

```
Seed  : 2
Base  : 0x00003ca9 (decimal 15529)
Limit : 500
```

Virtual Address	In Bounds	Translation, if yes
0: 0x00000039 (decimal: 57)	VALID	0x00003ce2 (decimal: 15586)
1: 0x00000056 (decimal: 86)	VALID	0x00003cff (decimal: 15615)
2: 0x00000357 (decimal: 855)	VIOLATION	-
3: 0x000002f1 (decimal: 753)	VIOLATION	-
4: 0x000002ad (decimal: 685)	VIOLATION	-

```
./relocation.py -s 3
```

```
Seed  : 3
Base  : 0x000022d4 (decimal 8916)
Limit : 316
```

Virtual Address	In Bounds	Translation, if yes
0: 0x0000017a (decimal: 378)	VIOLATION	-
1: 0x0000026a (decimal: 618)	VIOLATION	-
2: 0x00000280 (decimal: 640)	VIOLATION	-
3: 0x00000043 (decimal: 67)	VALID	0x00002317 (decimal: 8983)
4: 0x0000000d (decimal: 13)	VALID	0x000022e1 (decimal: 8929)

Part 2

```
./relocation.py -s 0 -n 10
```

Because of the VA 8: 0x000003a1 (decimal: 929), the limit must be greater than or equal to 930 (= 929+1) to ensure all the generated virtual addresses are within bounds.

Part 3

```
./relocation.py -s 1 -n 10 -l 100
```

```
Memory Size is 16K: 16384
Limit: 100
Maximum Base Address: 16384 - 100 = 16284
```

Part 4

```
./relocation.py -s <SEED_VALUE> -a 16k -p 128k
```

```
Address Space Size: 16K
Physical Memory Size: 128K
Seed Value: 1, 2, 3

Seed  : 1
Base  : 0x0001b1e2 (decimal 111074)
Limit : 4645
```

Virtual Address	In Bounds	Translation, if yes
0: 0x000030e1 (decimal: 12513)	VIOLATION	-

1: 0x00001053 (decimal: 4179)	VALID	0x0001c235 (decimal: 115253)
2: 0x00001fb5 (decimal: 8117)	VIOLATION	-
3: 0x00001cc4 (decimal: 7364)	VIOLATION	-
4: 0x000029b3 (decimal: 10675)	VIOLATION	-

```
Seed  : 2
Base  : 0x00001cf4 (decimal 7412)
Limit : 8011
```

Virtual Address	In Bounds	Translation, if yes
0: 0x0000056e (decimal: 1390)	VALID	0x00002262 (decimal: 8802)
1: 0x00003578 (decimal: 13688)	VIOLATION	-
2: 0x00002f1a (decimal: 12058)	VIOLATION	-
3: 0x00002adc (decimal: 10972)	VIOLATION	-
4: 0x000013b8 (decimal: 5048)	VALID	0x000030ac (decimal: 12460)

```
Seed  : 3
Base  : 0x000116a5 (decimal 71333)
Limit : 5070
```

Virtual Address	In Bounds	Translation, if yes
0: 0x000017ad (decimal: 6061)	VIOLATION	-
1: 0x000026a6 (decimal: 9894)	VIOLATION	-
2: 0x0000280b (decimal: 10251)	VIOLATION	-
3: 0x00000431 (decimal: 1073)	VALID	0x00011ad6 (decimal: 72406)
4: 0x000000d7 (decimal: 215)	VALID	0x0001177c (decimal: 71548)

```
./relocation.py -s 0 -n 10 -a 16k -p 128k
```

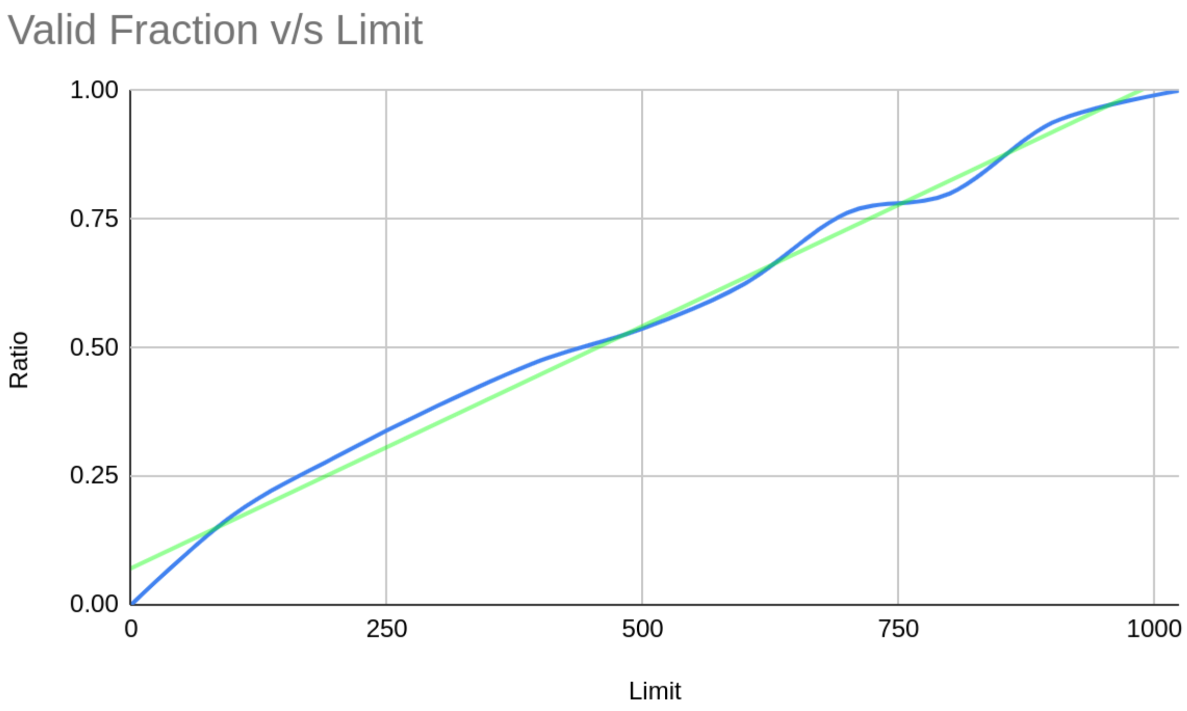
Because of VA 8: 0x00003a1e (decimal: 14878), the limit must be greater than or equal to 14879 (= 14878+1) to ensure all the generated virtual addresses are within bounds.

5

```
 ./relocation.py -s 0 -n 10 -l 100 -a 16k -p 128k
```

```
Memory Size is 128K: 131072
Limit: 100
Maximum Base Address: 131072 - 100 = 130972
```

Part 5



For the data used for experiment: Lab 07

Question 2

Part 1

Seed: 0, 1, 2
Address Space Size: 128
Physical Memory Size: 512

```
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0
```

Virtual Address	In Bounds	Translation, if yes
0: 0x0000006c (decimal: 108)	VALID	0x000001ec (decimal: 492)
1: 0x00000061 (decimal: 97)	VIOLATION	-
2: 0x00000035 (decimal: 53)	VIOLATION	-
3: 0x00000021 (decimal: 33)	VIOLATION	-
4: 0x00000041 (decimal: 65)	VIOLATION	-

```
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1
```

Virtual Address	In Bounds	Translation, if yes
0: 0x00000011 (decimal: 17)	VALID	0x00000011 (decimal: 17)
1: 0x0000006c (decimal: 108)	VALID	0x000001ec (decimal: 492)
2: 0x00000061 (decimal: 97)	VIOLATION	-
3: 0x00000020 (decimal: 32)	VIOLATION	-
4: 0x0000003f (decimal: 63)	VIOLATION	-

```
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2
```

Virtual Address	In Bounds	Translation, if yes
0: 0x0000007a (decimal: 122)	VALID	0x000001fa (decimal: 506)
1: 0x00000079 (decimal: 121)	VALID	0x000001f9 (decimal: 505)

2: 0x00000007 (decimal: 7)	VALID	0x00000007 (decimal: 7)
3: 0x0000000a (decimal: 10)	VALID	0x0000000a (decimal: 10)
4: 0x0000006a (decimal: 106)	VIOLATION	-

Part 2

Highest Legal Virtual Address in SEG0 : 19
Lowest Legal Virtual Address in SEG1 : 108
Highest Illegal Virtual Address in entire space : 107
Lowest Illegal Virtual Address in entire space : 20

```
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -A <ADDRESS>
```

We can check this by putting 19, 20, 107 and 108 in place of the address in the above command.

Part 3

We need all the addresses between 3 to 13 to be registered as violations, so I would set in the following manner:
BASE_SEG0 : 1
LIMIT_SEG0 : 2

BASE_SEG1 : 15
LIMIT_SEG1 : 2
So my command will be the following:

```
./segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 0 --l0
2 --b1 15 --l1 2
```

Part 4

The sum of both limits must be 90% of the address space. Also:
BASE_0 + LIMIT_0 < BASE_1 - LIMIT_1
Or, BASE_1 - BASE_0 > LIMIT_0 + LIMIT_1

Part 5

We can simply set the limit to zero.

Question 3

```
 ./paging-linear-size.py -v 32 -p 256 -e 4
```

```
VSIZE       : 32
PTESIZE     : 256 bytes
PAGESIZE    : 4
Offset Bits: log(256) = 8
Virtual Page Bits: 32 - 8 = 24
Size of page table: (2 ^ 24) * 4 = 67108864 bytes
```

```
 ./paging-linear-size.py -v 64 -p 512 -e 4
```

```
VSIZE       : 64
PTESIZE     : 512 bytes
PAGESIZE    : 4
Offset Bits: log(512) = 9
Virtual Page Bits: 64 - 9 = 55
Size of page table: (2 ^ 55) * 4 = 144115188075855870 bytes
```

```
 ./paging-linear-size.py -v 32 -p 16348 -e 8
```

```
VSIZE       : 32
PTESIZE     : 16348 bytes
PAGESIZE    : 8
Offset Bits: log(16348) = 14
Virtual Page Bits: 32 - 14 = 18
Size of page table: (2 ^ 18) * 8 = 2097152 bytes
```

```
 ./paging-linear-size.py -v 40 -p 2048 -e 20
```

```
VSIZE       : 40
PTESIZE     : 2048 bytes
PAGESIZE    : 20
Offset Bits: log(2048) = 11
Virtual Page Bits: 40 - 11 = 29
Size of page table: (2 ^ 29) * 20 = 10737418240 bytes
```

Question 4

Part 1

```
 PAGE SIZE: 1K
 ADDRESS SPACE SIZE: 1M, 2M, 4M
 PHYSICAL MEMORY SIZE: 512M
```

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

```
Size: 1024
```

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

```
Size: 2048
```

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

```
Size: 4096
```

The size of page table increases linearly to address space size.

```
PAGE SIZE: 1K, 2K, 4K
ADDRESS SPACE SIZE: 1M
PHYSICAL MEMORY SIZE: 512M
```

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

```
Size: 1024
```

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

```
Size: 512
```

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

```
Size: 256
```

The size of page table decreases linearly to page size.

Big pages leave a lot of unused memory as processes can only fill a fraction of the space.

Part 2

As expected, the size of the page table remains the same, i.e. 16. Increasing the percentage of allocated pages only increases the number of pages filled in the table.

```
PAGE SIZE: 1K
ADDRESS SPACE SIZE: 16K
PHYSICAL MEMORY SIZE: 32K
For u = 0, 0 out of 5 pages were allocated.
For u = 25, 1 out of 5 pages were allocated.
For u = 50, 3 out of 5 pages were allocated.
For u = 75, 5 out of 5 pages were allocated.
For u = 100, 5 out of 5 pages were allocated.
```

Part 3

```
PAGE SIZE: 8
ADDRESS SPACE SIZE: 32
PHYSICAL MEMORY SIZE: 1024
```

The first parameter is very unrealistic and extremely useless in for any real-world use case.

Part 4

- Physical Memory Size must be positive, as otherwise, there will not be any space to store the data for the CPU.
- Address Space Size must be positive. Otherwise, no space will be there to store the pages,
- Physical Memory Size should be greater than or equal to Address Memory Size. Otherwise, No simulation will be possible, as there will not be much space to fill the pages in the physical memory.
- If the Address Memory Size is greater than or equal to 1GB, the simulator will throw an error as our simulator can not handle such a large simulation.
- Physical Memory must be multiple of page size, otherwise, the simulator will throw an error as an integral number of pages can not be inside the physical memory.
- Address Memory must be a multiple of the page size. Same reason as above.
- Page Size and Address Space size must be a multiple of 2.