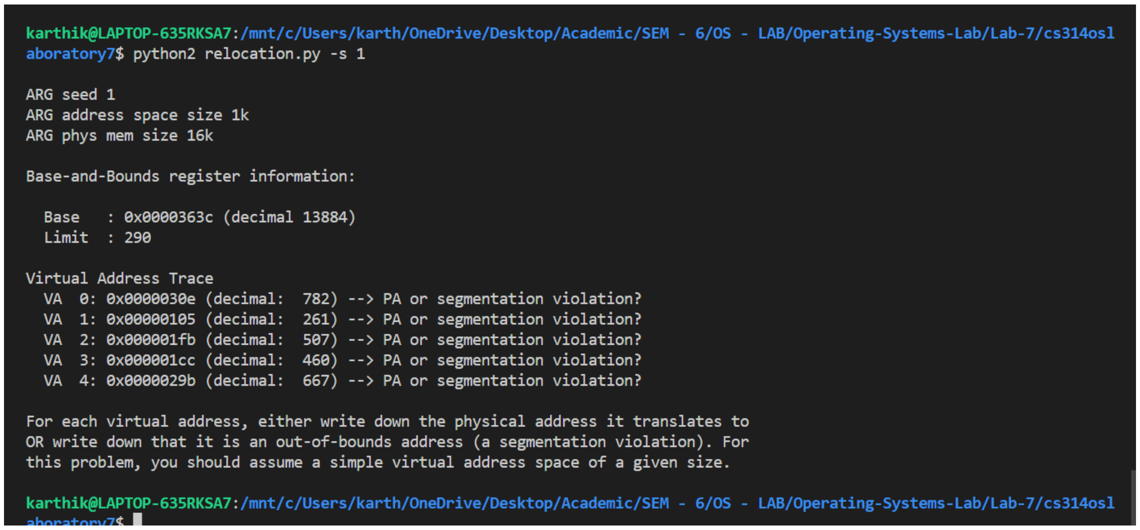
CS 314 Operating Systems Lab

Assignment 7

- Karthik K (200010024)

Q1

1. After Running the program relocation.py the output of that is shown below. Seed : 1

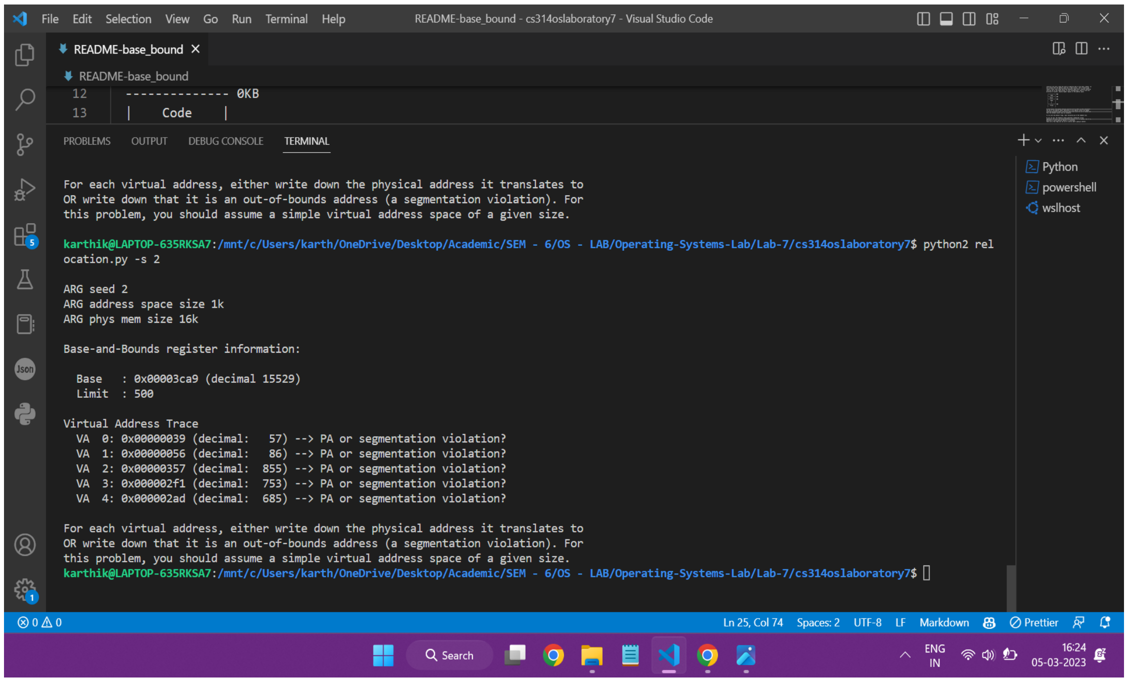


```
karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$ python2 relocation.py -s 1

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x0000363c (decimal 13884)
  Limit  : 290

Virtual Address Trace
  VA  0: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  1: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  2: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  3: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  4: 0x0000029b (decimal:  667) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$
```

Virtual Address Trace
  VA 0: 0x0000030e (decimal: 782) --> segmentation violation
  VA 1: 0x00000105 (decimal: 261) -->   0x3741
  VA 2: 0x000001fb (decimal: 507) --> segmentation violation
  VA 3: 0x000001cc (decimal: 460) --> segmentation violation
  VA 4: 0x0000029b (decimal: 667) --> segmentation violation

Bound is at 290

After Running the program relocation.py the output of that is shown below. Seed : 2



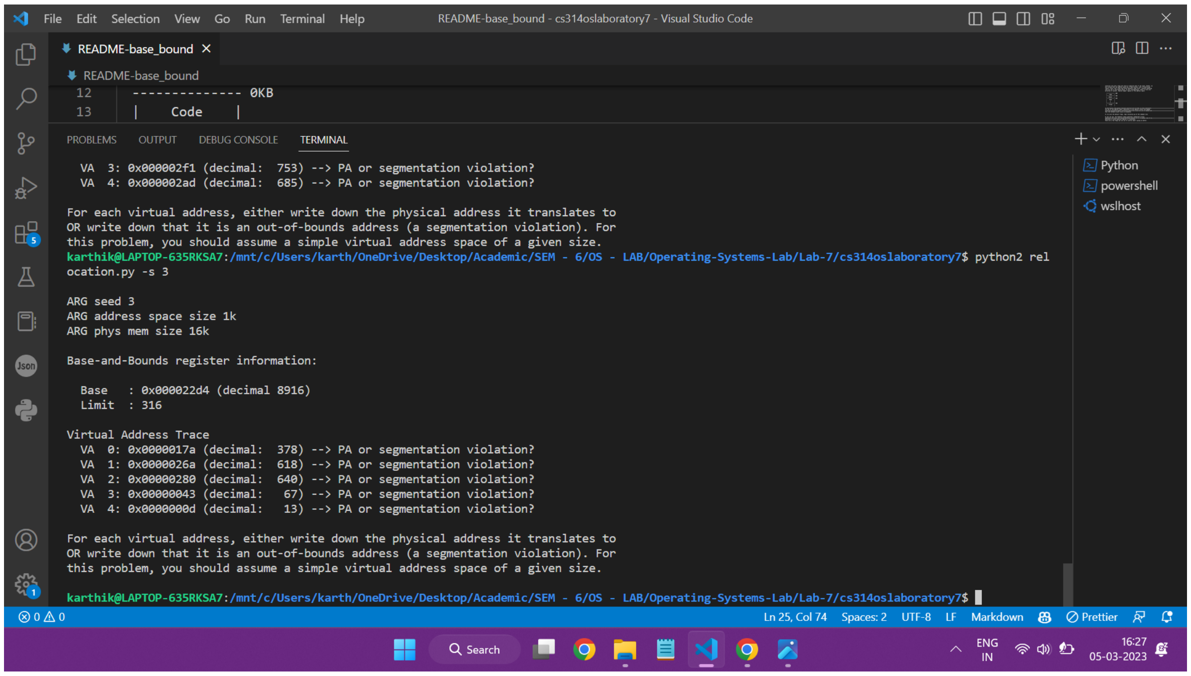
```
-------------- 0KB
|     Code     |
```

```
For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314oslaboratory7$ python2 rel
ocation.py -s 2

ARG seed 2
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003ca9 (decimal 15529)
  Limit  : 500

Virtual Address Trace
  VA  0: 0x00000039 (decimal:   57) --> PA or segmentation violation?
  VA  1: 0x00000056 (decimal:   86) --> PA or segmentation violation?
  VA  2: 0x00000357 (decimal:  855) --> PA or segmentation violation?
  VA  3: 0x000002f1 (decimal:  753) --> PA or segmentation violation?
  VA  4: 0x000002ad (decimal:  685) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314oslaboratory7$
```

Virtual Address Trace
VA 0: 0x00000039 (decimal: 57) --> 0x3CE2
VA 1: 0x00000056 (decimal: 86) --> 0x3CE2
VA 2: 0x00000357 (decimal: 855) -->segmentation violation
VA 3: 0x000002f1 (decimal: 753) --> segmentation violation
VA 4: 0x000002ad (decimal: 685) -->segmentation violation

Bound is at 500

After Running the program relocation.py the output of that is shown below. Seed : 3

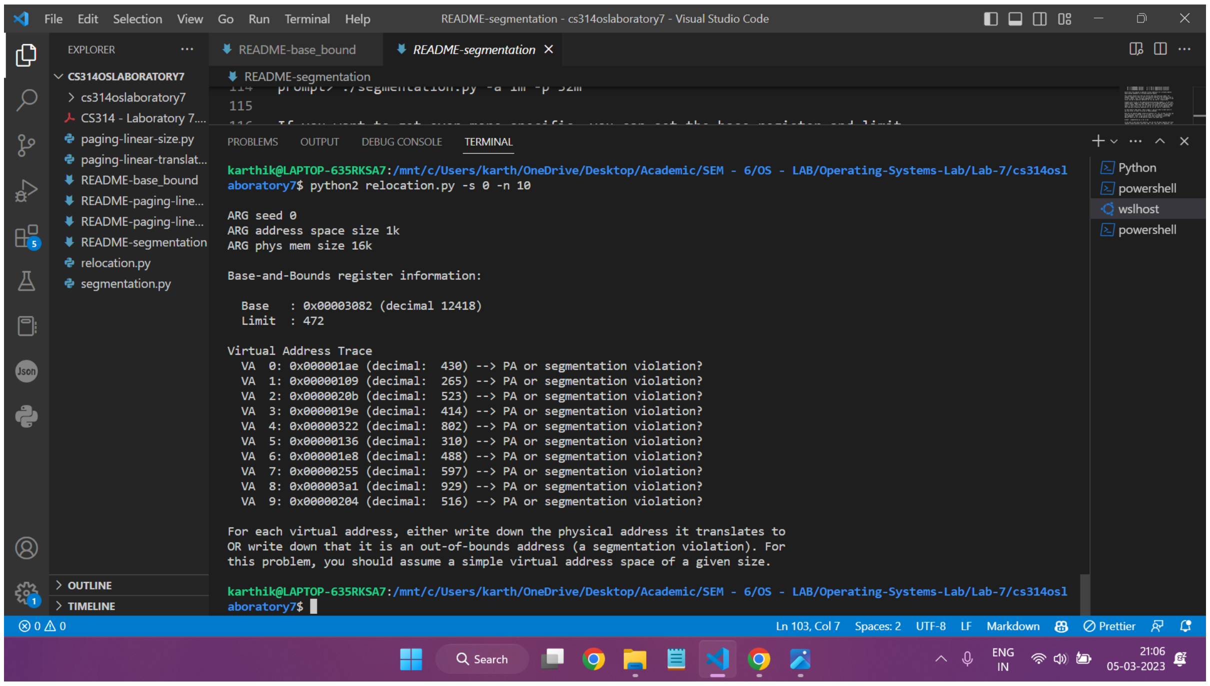


```
  VA  3: 0x000002f1 (decimal:  753) --> PA or segmentation violation?
  VA  4: 0x000002ad (decimal:  685) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314oslaboratory7$ python2 rel
ocation.py -s 3

ARG seed 3
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x000022d4 (decimal 8916)
  Limit  : 316

Virtual Address Trace
  VA  0: 0x0000017a (decimal:  378) --> PA or segmentation violation?
  VA  1: 0x0000026a (decimal:  618) --> PA or segmentation violation?
  VA  2: 0x00000280 (decimal:  640) --> PA or segmentation violation?
  VA  3: 0x00000043 (decimal:   67) --> PA or segmentation violation?
  VA  4: 0x0000000d (decimal:   13) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314oslaboratory7$
```

Virtual Address Trace
  VA 0: 0x0000017a (decimal: 378) --> segmentation violation
  VA 1: 0x0000026a (decimal: 618) --> segmentation violation
  VA 2: 0x00000280 (decimal: 640) --> segmentation violation
  VA 3: 0x00000043 (decimal: 67) --> 0x2317
  VA 4: 0x0000000d (decimal: 13) → 0x22E1

Bound is at 316

2. The output of the program with -s 0 -n 10 is shown below.
   Current Bound is 472

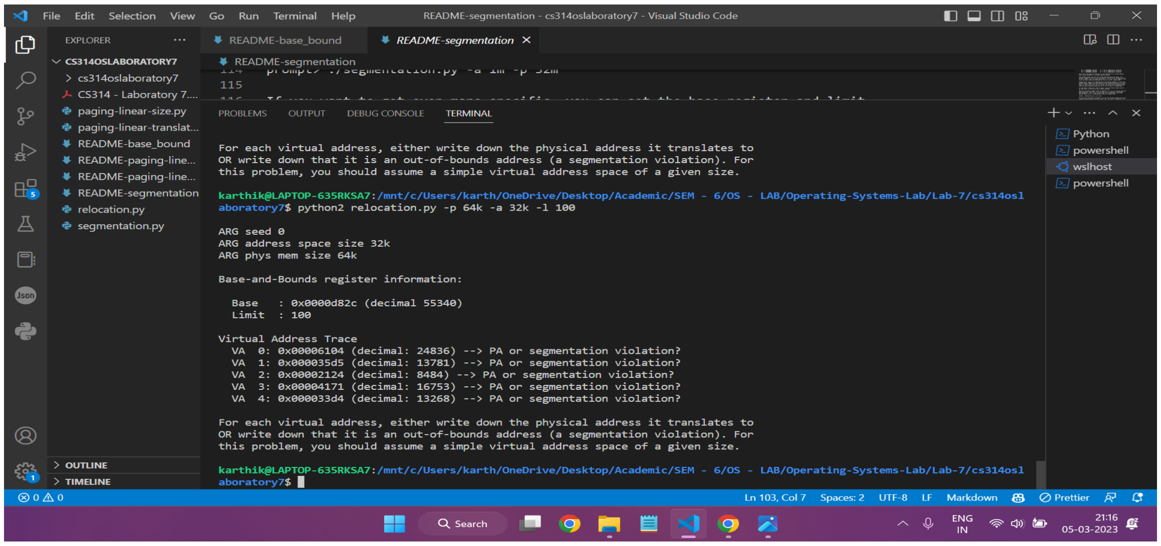


```
karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$ python2 relocation.py -s 0 -n 10

ARG seed 0
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003082 (decimal 12418)
  Limit  : 472

Virtual Address Trace
  VA  0: 0x000001ae (decimal:  430) --> PA or segmentation violation?
  VA  1: 0x00000109 (decimal:  265) --> PA or segmentation violation?
  VA  2: 0x0000020b (decimal:  523) --> PA or segmentation violation?
  VA  3: 0x0000019e (decimal:  414) --> PA or segmentation violation?
  VA  4: 0x00000322 (decimal:  802) --> PA or segmentation violation?
  VA  5: 0x00000136 (decimal:  310) --> PA or segmentation violation?
  VA  6: 0x000001e8 (decimal:  488) --> PA or segmentation violation?
  VA  7: 0x00000255 (decimal:  597) --> PA or segmentation violation?
  VA  8: 0x000003a1 (decimal:  929) --> PA or segmentation violation?
  VA  9: 0x00000204 (decimal:  516) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$ 
```

We can see in the image above that the virtual addresses are listed above. If we want to make sure that no address is out of bound then we need to set the bound register to maximum virtual address +1 i.e 929+1 = 930 according to the output shown above.

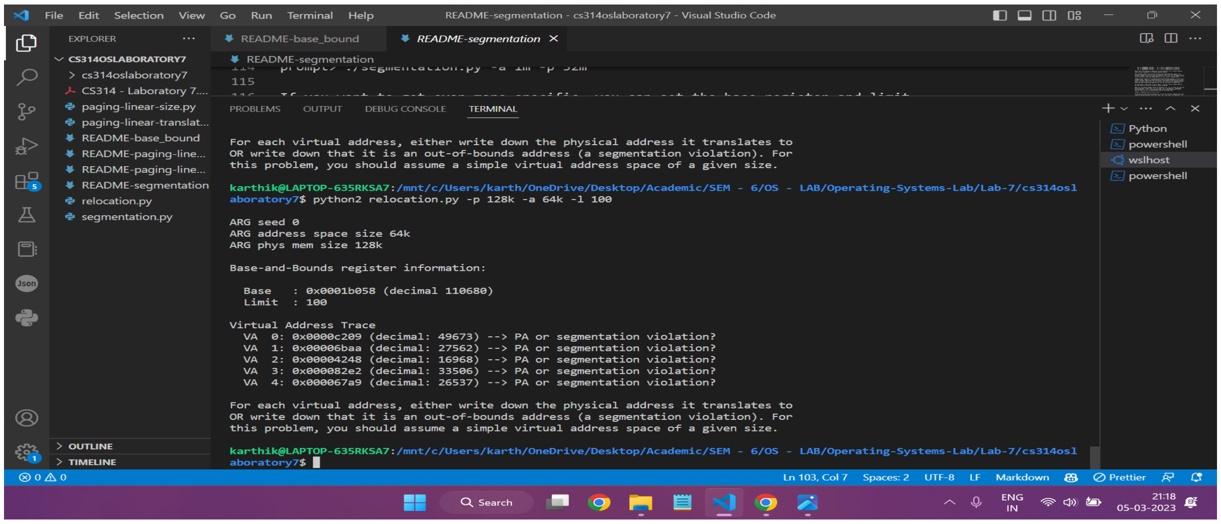
3. The ARG physical memory size is 16 kB. So if we to store the valid virtual address in the physical memory safely then the base value can go up to 16*1024 - 100 = 16384 - 100 = 16284

4. I ran the program with 64 KB physical memory and 32 KB virtual memory. For making sure that all the virtual addresses are in valid range we need to set the bound to maxAddress + 1 = 24836+1 = 24837. Here the maximum base value can be 64*1024 - 100 = 65436 bytes.



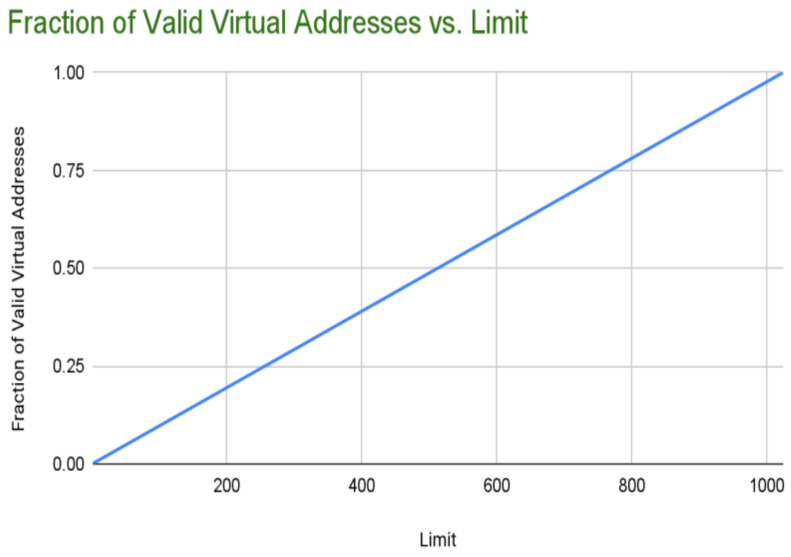
```
For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$ python2 relocation.py -p 64k -a 32k -l 100

ARG seed 0
ARG address space size 32k
ARG phys mem size 64k

Base-and-Bounds register information:

  Base   : 0x0000d82c (decimal 55340)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00006104 (decimal: 24836) --> PA or segmentation violation?
  VA  1: 0x000035d5 (decimal: 13781) --> PA or segmentation violation?
  VA  2: 0x00002124 (decimal: 8484) --> PA or segmentation violation?
  VA  3: 0x00004171 (decimal: 16753) --> PA or segmentation violation?
  VA  4: 0x000033d4 (decimal: 13268) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$
```

When I ran the program with 128 KB physical memory and 64 KB virtual memory. For making sure that all the virtual addresses are in valid range we need to set the bound to maxAddress + 1 = 49673+1 = 49674. Here the maximum base value can be 128*1024 - 100 = 130972 bytes.



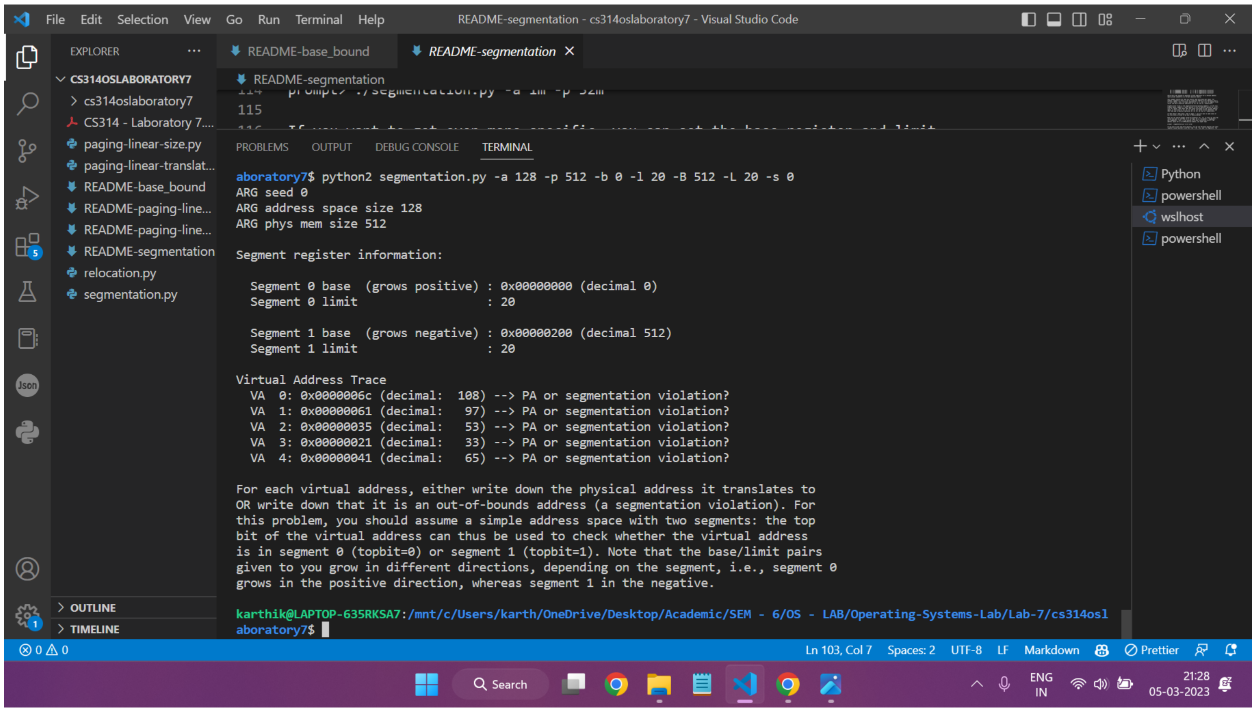
```
For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$ python2 relocation.py -p 128k -a 64k -l 100

ARG seed 0
ARG address space size 64k
ARG phys mem size 128k

Base-and-Bounds register information:

  Base   : 0x0001b058 (decimal 110680)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x0000c209 (decimal: 49673) --> PA or segmentation violation?
  VA  1: 0x00006baa (decimal: 27562) --> PA or segmentation violation?
  VA  2: 0x00004248 (decimal: 16968) --> PA or segmentation violation?
  VA  3: 0x000082e2 (decimal: 33506) --> PA or segmentation violation?
  VA  4: 0x000067a9 (decimal: 26537) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$
```

5. The fraction of valid virtual memory addresses will be
   $F(limit) = (limit) / (Virtual\ Address\ Space)$
   Virtual Address Space for the below graph is 1 kB.



Q2)

1. After running the programs with the parameters specified here are the outputs of those.



```
aboratory7$ python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> PA or segmentation violation?
  VA  1: 0x00000061 (decimal:   97) --> PA or segmentation violation?
  VA  2: 0x00000035 (decimal:   53) --> PA or segmentation violation?
  VA  3: 0x00000021 (decimal:   33) --> PA or segmentation violation?
  VA  4: 0x00000041 (decimal:   65) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment 0
grows in the positive direction, whereas segment 1 in the negative.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$
```

Seed 0
VA 0: 0x0000006c (decimal: 108)
(108) base 10 → (1101100) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 101100 (44) - 1000000 (64) = -20
|-20| <= limit of segment 1(20), So it’s a valid address
Physical address = Base + Offset = 512 - 20 = 492

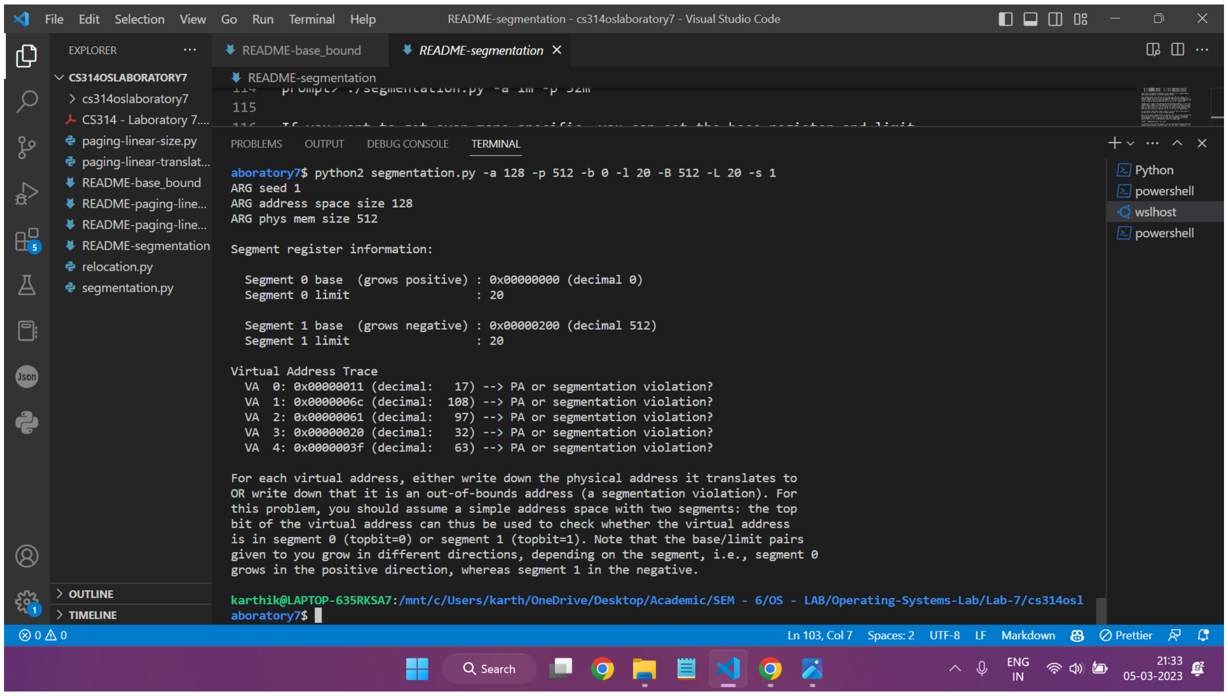
VA 1: 0x00000061 (decimal: 97) (97)
base 10 → (1100001) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 100001 (33) - 1000000 (64) = -31
|-31| <= limit of segment 1 (20), So it’s a invalid address So,
Segmentation Fault

VA 2: 0x00000035 (decimal: 53)
(53) base 10 → (0110101) base 2
The segment bit is 0 i.e means it belongs to segment 0.
Offset = 110101 (53) |53| > limit of segment 0 (20), So it’s a invalid
address So, Segmentation Fault

VA 3: 0x00000021 (decimal: 33)
(33) base 10 → (0100001) base 2
The segment bit is 0 i.e means it belongs to segment 0.
Offset = 100001 (33) |33| > limit of segment 0 (20), So it’s a invalid
address So, Segmentation Fault

VA 4: 0x00000041 (decimal: 65)
(65) base 10 → (1000001) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 000001 (1) - 1000000 (64) = -63
|-63| > limit of segment 1 (20), So it’s a invalid address So,
Segmentation Fault

Seed 1



```
aboratory7$ python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> PA or segmentation violation?
  VA  1: 0x0000006c (decimal:  108) --> PA or segmentation violation?
  VA  2: 0x00000061 (decimal:   97) --> PA or segmentation violation?
  VA  3: 0x00000020 (decimal:   32) --> PA or segmentation violation?
  VA  4: 0x0000003f (decimal:   63) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment 0
grows in the positive direction, whereas segment 1 in the negative.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$
```

VA 0: 0x00000011 (decimal: 17)
 (17) base 10 → (0010001) base 2
The segment bit is 0 i.e means it belongs to segment 0.
 Offset = 010001 (17) |17| <= limit of segment 0(20), So it's a valid address Physical address = Base + Offset = 0 + 17 = 17

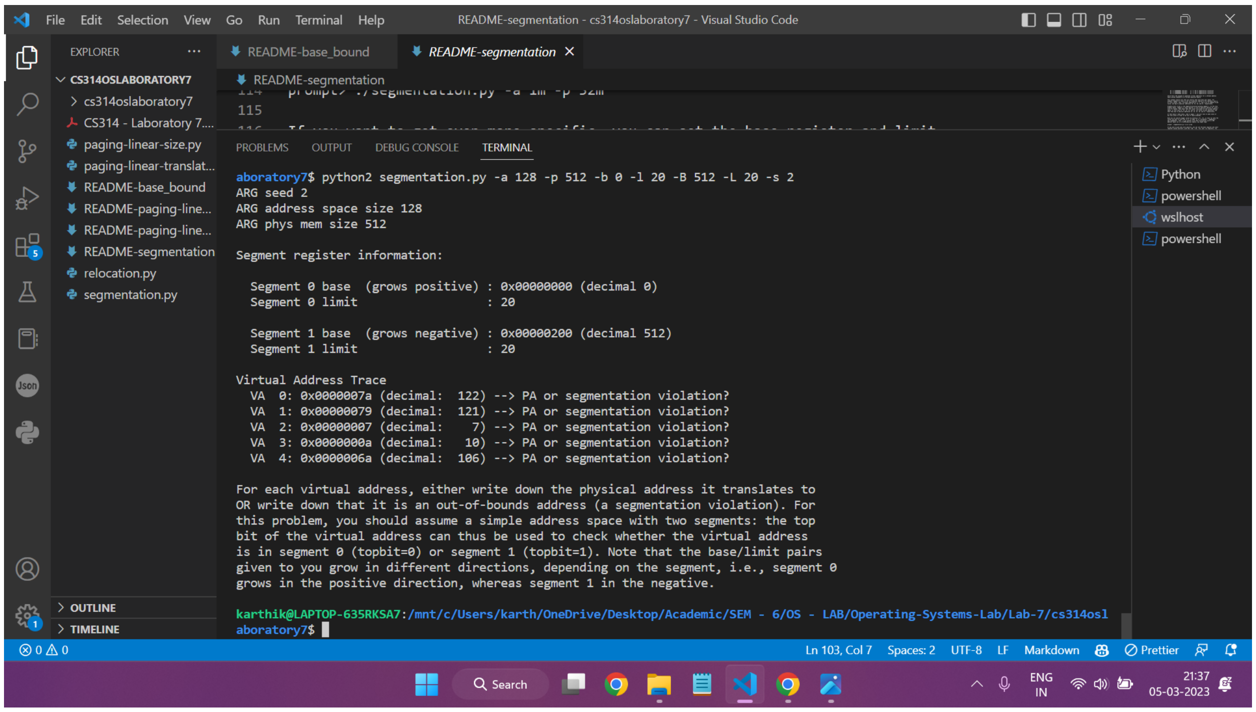
VA 1: 0x0000006c (decimal: 108)
 (108) base 10 → (1101100) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 101100 (44) - 1000000 (64) = -20
 |-20| <= limit of segment 1(20), So it's a valid address Physical address = Base + Offset = 512 - 20 = 492

VA 2: 0x00000061 (decimal: 97)
(97) base 10 → (1100001) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 100001 (33) - 1000000 (64) = -31 |-31| <= limit of segment 1 (20), So it's a invalid address So, Segmentation Fault

VA 3: 0x00000020 (decimal: 32)
(32) base 10 → (0100000) base 2
The segment bit is 0 i.e means it belongs to segment 0.
 Offset = 100000 (32) |32| > limit of segment 0 (20), So it's a invalid address So, Segmentation Fault

VA 4: 0x0000003f (decimal: 63)
(63) base 10 → (0111111) base 2
The segment bit is 0 i.e means it belongs to segment 0.
 Offset = 111111 (63) |63| > limit of segment 0 (20), So it's a invalid address So, Segmentation Fault

Seed 2



```
aboratory7$ python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2
ARG seed 2
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> PA or segmentation violation?
  VA  1: 0x00000079 (decimal:  121) --> PA or segmentation violation?
  VA  2: 0x00000007 (decimal:    7) --> PA or segmentation violation?
  VA  3: 0x0000000a (decimal:   10) --> PA or segmentation violation?
  VA  4: 0x0000006a (decimal:  106) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment 0
grows in the positive direction, whereas segment 1 in the negative.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$
```

VA 0: 0x0000007a (decimal: 122)
 (122) base 10 → (1111010) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 111010 (58) - 1000000 (64) = -6

|-6| <= limit of segment 1(20), So it's a valid address Physical address = Base + Offset = 512 - 6 = 506

VA 1: 0x00000079 (decimal: 121)
(121) base 10 → (1111001) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 111001 (57) - 1000000 (64) = -7
|-7| <= limit of segment 1(20), So it's a valid address Physical address = Base + Offset = 512 - 7 = 505

VA 2: 0x00000007 (decimal: 7)
(7) base 10 → (0000111) base 2
The segment bit is 0 i.e means it belongs to segment 0.
Offset = 000111 (7) |7| <= limit of segment 0(20), So it's a valid address Physical address = Base + Offset = 0 + 7 = 7

VA 3: 0x0000000a (decimal: 10)
(10) base 10 → (0001010) base 2
The segment bit is 0 i.e means it belongs to segment 0.
Offset = 001010 (10) |10| <= limit of segment 0(20), So it's a valid address Physical address = Base + Offset = 0 + 10 = 10

VA 4: 0x0000006a (decimal: 106)
(106) base 10 → (1101010) base 2
The segment bit is 1 i.e means it belongs to segment 1.
Offset = 101010 (42) - 1000000 (64) = -22 |-22| > limit of segment 1(20), So it's a invalid address So, Segmentation Fault

2. The Highest legal virtual address in segment 0 is 19.
   The lowest legal virtual address in segment 1 is 108.
   The lowest illegal virtual address in the whole address space is 20.
   We can use this command to test it.

```
python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2 -A 20
```

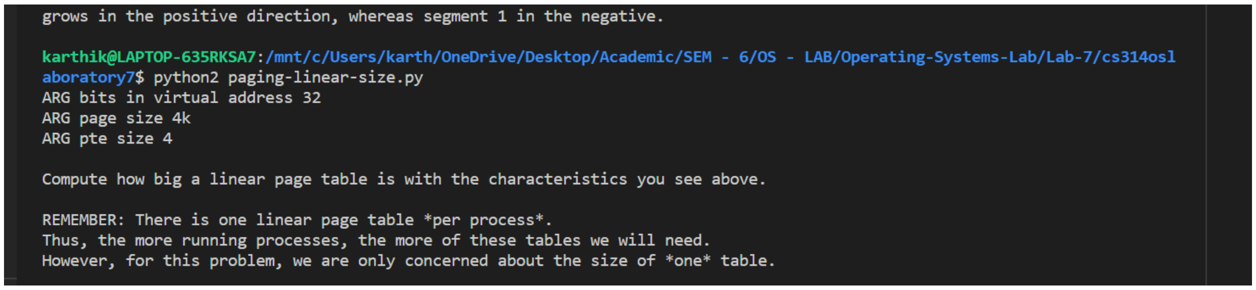
The highest illegal virtual address in the whole address space is 107. We can use this command to test it.

python2 segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2 -A 107

3. In this we have to have two valid addresses in the starting and two valid addresses at the end of the address space. The physical address space is 128 bytes. So the base of segment 0 is 0 and the bound of segment 0 is 2. The base of segment 1 is 128 and the bound of segment 1 is 2.
python2 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 128 --l1 2
4. We need 90% of the address space in either of the segments. So let the address space be 512 KB then we can have the bound for both the segments at 230 KB (~512*0.9/2). Base for segment 0 is 0 and base for segment 1 is 1024 KB if the physical address space is 1024 KB
python2 segmentation.py -a 512 -p 1024 --b0 0 --l0 230 --b1 1024 --l1 230 -n 100
The parameters -bo, -lo, -b1, -l1 -p, -a are important.
5. Yes, if both the bounds are equal to zero and bases of the segments are equal.

Q3)

The output of the program ‘paging-linear-size.py’ is shown below



```
grows in the positive direction, whereas segment 1 in the negative.

karthik@LAPTOP-635RKSA7:/mnt/c/Users/karth/OneDrive/Desktop/Academic/SEM - 6/OS - LAB/Operating-Systems-Lab/Lab-7/cs314osl
aboratory7$ python2 paging-linear-size.py
ARG bits in virtual address 32
ARG page size 4k
ARG pte size 4

Compute how big a linear page table is with the characteristics you see above.

REMEMBER: There is one linear page table *per process*.
Thus, the more running processes, the more of these tables we will need.
However, for this problem, we are only concerned about the size of *one* table.
```

The page size is 4KB = 4096 bytes. This requires 12 bits for offset which leaves 20 bits for representing the page number as the virtual address is 32 bits. The number of pages possible to represent with 20 bits is 2^20=1048576. It's given that each entry in the page table requires 4 bytes of memory. So, the maximum size of this linear page table can be 4 *1048576 = 4194304 bytes = 4MB.

Q4)

1. The page table contains an entry for each page created for that particular process. The number of entries in the page table is equal to the number of pages present which is $(Address\ Space\ Size)$ / $(Page\ Size)$ So if the address space size increases then the number of pages will increase and if page size increases the number of pages will Decrease. We should always have a small page size as creating large pages will occupy a lot of physical memory space which may not be even used by the process. Most processes will only use less memory when compared to the total space available in the physical memory.
   After running the program with these parameters this is the number of entries in the page table. -a sets the address space size and -P sets the Page size.

   -P 1k -a 1m -p 512m -v -n 0 - 1024 KB / 1 KB = 1024 Pages
   -P 1k -a 2m -p 512m -v -n 0 - 2048 KB / 1 KB = 2048 Pages
   -P 1k -a 4m -p 512m -v -n 0 - 4096 KB / 1 KB = 4096 Pages

   We can see as the address space size increases the number of entries in the page table also increases. After running the program with these parameters this is the number of entries in the page table. -a sets the address space size and -P sets the page size.

   -P 1k -a 1m -p 512m -v -n 0 - 1024 KB / 1 KB = 1024 Pages
   -P 2k -a 1m -p 512m -v -n 0 - 1024 KB / 2 KB = 512 Pages
   -P 4k -a 1m -p 512m -v -n 0 - 1024 KB / 4 KB = 256 Pages

   We can see as the page size increases the number of entries in the page table decreases.

2. The -u tag is used to set the percentage of valid physical addresses. The page table contains virtual addresses to physical address mapping. All pages need not be mapped to in the physical memory all the time. So in case if a page is not mapped then the address is marked as invalid. The first bit in the physical address tells us about that. So with -u tag we can set the percentage of pages mapped to a valid physical address. So as we increase the percentage we can see that the number of valid memory accesses increases and free space decreases.

   These are the results I observed when running the program with these parameters.

   -P 1k -a 16k -p 32k -v -u 0 - No of successful memory accesses = 0
   -P 1k -a 16k -p 32k -v -u 25 - No of successful memory accesses = 1
   -P 1k -a 16k -p 32k -v -u 50 - No of successful memory accesses = 3
   -P 1k -a 16k -p 32k -v -u 75 - No of successful memory accesses = 5
   -P 1k -a 16k -p 32k -v -u 100 - No of successful memory accesses = 5

3. These parameters in the first case looked unrealistic as the values are too small to be true. An address space of 32 bytes is very rare as an int in a c program will itself consume 4 bytes and whole address space will be consumed if 8 variables are used. This is very rare in modern day processes.

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

4. The given program raises an error in the following conditions:
   1. Physical Memory size is negative
   2. Address Space size is negative
   3. Page Size is negative
   4. Physical Memory is not a multiple of page size
   5. Address Space Size is not a multiple of page size
   6. Address Space Size is greater than Physical Memory Size
   7. Page Size is greater than Address Space
   8. Page Size is greater than Physical Memory Size
   9. Address Space size is not a power of 2