Quiz 4

Your Name: ________________________

1. Virtual address space vs physical memory space, which one is bigger? (5 points)

2. Here is a little program that prints out the locations of the main() routine (where code lives), the value of a heap-allocated value returned from malloc(), and the location of an integer on the stack:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("location of code : %p\n", main);
    printf("location of heap : %p\n", malloc(100e6));
    int x = 3;
    printf("location of stack: %p\n", &x);
    return 0;
}

When run on a 64-bit Mac, we get the following output:
location of code : 0x1095afe50
location of heap : 0x1096008c0
location of stack: 0x7fff691aea64
```

In the output of the above program, is this address - 0x1095afe50, a virtual address or a physical address? (5 points)

In the output of the above program, is this address - 0x1096008c0, a virtual address or a physical address? (5 points)

In the output of the above program, is this address - 0x7fff691aea64, a virtual address or a physical address? (5 points)

3. **Address Translation: Segmentation**. (20 points) A simulation trace is generated below using the tool segmentation.py.

```
ARG address space size 128
ARG phys mem size 256

Visually, the address space looks like this:

 --------------- virtual address 0
  |    seg0     |
  |             |
  |             |
  |-------------|
  |             |
  |             |
  |             |
  |             |
  |(unallocated)|
  |             |
  |             |
  |             |
  |-------------|
  |             |
  |    seg1     |
  |-------------| virtual address max (size of address space)

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 40

  Segment 1 base  (grows negative) : 0x00000100 (decimal 256)
  Segment 1 limit                  : 40

Virtual Address Trace
  VA  0: 0x00000065 (decimal:  101) --> PA or segmentation violation?
  VA  1: 0x00000069 (decimal:  105) --> PA or segmentation violation?
  VA  2: 0x0000003e (decimal:   62) --> PA or segmentation violation?
  VA  3: 0x00000021 (decimal:   33) --> 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.

Answer:

  VA  0: 0x00000065 (decimal:  101) --> VALID in SEG1: 0x000000e5 (decimal:  229)
  VA  1: 0x00000069 (decimal:  105) --> VALID in SEG1: 0x000000e9 (decimal:  233)
  VA  2: 0x0000003e (decimal:   62) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000021 (decimal:   33) --> VALID in SEG0: 0x00000021 (decimal:   33)
```

4. **Address Translation: Paging**. (40 points)

   A simulation trace is generated below using the tool paging-linear-translate.py.

```
ARG address space size 8k
ARG phys mem size 32k
ARG page size 1k

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.

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

Virtual Address Trace
  VA 0x00000803 (decimal:     2051) --> PA or invalid address?
  VA 0x00001d1b (decimal:     7451) --> PA or invalid address?
  VA 0x000019ec (decimal:     6636) --> PA or invalid address?
  VA 0x00001cde (decimal:     7390) --> 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).

Answer:

  VA 0x00000803 (decimal:     2051) -->  Invalid (VPN 2 not valid)
  VA 0x00001d1b (decimal:     7451) --> 00004d1c (decimal    19739) [VPN 7]
  VA 0x000019ec (decimal:     6636) --> 000075ed (decimal    30188) [VPN 6]
  VA 0x00001cde (decimal:     7390) --> 00004cde (decimal    19678) [VPN 7]
```

5. **Address Translation: Paging**. (20 points) A simulation trace is generated below using the tool paging-linear-translate.py.

```
ARG address space size 128
ARG phys mem size 512
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]   0x8000002f
  [       1]   0x8000003c
  [       2]   0x8000003b
  [       3]   0x00000000
  [       4]   0x00000000
  [       5]   0x80000029
  [       6]   0x80000007
  [       7]   0x00000000
  [       8]   0x00000000
  [       9]   0x80000024
  [      10]   0x00000000
  [      11]   0x00000000
  [      12]   0x00000000
  [      13]   0x80000031
  [      14]   0x00000000
  [      15]   0x80000008

Virtual Address Trace
  VA 0x0000004f (decimal:       79) --> PA or invalid address?
  VA 0x0000001c (decimal:       28) --> 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).

Answer:

  VA 0x0000004f (decimal:       79) --> 00000127 (decimal      295) [VPN 9]
  VA 0x0000001c (decimal:       28) -->  Invalid (VPN 3 not valid)
```