Chapter 15

1. **1. Run with seeds 1, 2, and 3, and compute whether each virtual address generated by the process is in or out of bounds. If in bounds, compute the translation.**

-s 1: only VA1 (261) is valid, translated address 14145
-s 2: VA0 and VA1 are valid, translated address 15586, 15615
-s 3: VA3 and VA4 are valid, translated address 8983,8929

2. Run with these flags: -s 0 -n 10. What value do you have set -l (the bounds register) to in order to ensure that all the generated virtual addresses are within bounds?
Bound needs to be greater than 929, minimum 930.

3. Run with these flags: -s 1 -n 10 -l 100. What is the maximum value that base can be set to, such that the address space still fits into physical memory in its entirety?
phys mem size is 16k (16 * 1024), limit(bound) is 100, so the max value base can be is 16 * 1024 – 100 = 16284

4. Run some of the same problems above, but with larger address spaces (-a) and physical memories (-p).
5. What fraction of randomly-generated virtual addresses are valid, as a function of the value of the bounds register? Make a graph from running with different random seeds, with limit values ranging from 0 up to the maximum size of the address space.

Chapter 16

1. **First let's use a tiny address space to translate some addresses. Here's a simple set of parameters with a few different random seeds; can you translate the addresses? segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0**

   -s 0: Only VA0 is valid, translated address: 512 – (128-108) = 492
   -s 1: VA0 and VA1 are valid, translated address: 0+17 = 17, 512 – (128-108) = 492
   -s 2:
   V0: 512 – (128 -122) = 506
   V1: 512 – (128 – 121) = 505
   V2: 0+ 7 = 17
   V3: 0+ 10 = 10

   2. Now, let's see if we understand this tiny address space we've constructed (using the parameters from the question above). What is the highest legal virtual address in segment 0? What about the lowest legal virtual address in segment 1? What are the lowest and highest illegal addresses in this entire address space? Finally, how would you run segmentation.py with the -A flag to test if you are right?
   highest legal virtual address in segment 0: 19
   lowest legal virtual address in segment 1: 108
   lowest illegal addresses in this entire address space: 0+19+1 = 20

highest illegal addresses in this entire address space: 512-20-1 = 491

Can be tested by
python .\segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -A 19 -c
python .\segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -A 20 -c (violation)
python .\segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -A 108 -c
python .\segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -A 107 -c (violation)

3. Let's say we have a tiny 16-byte address space in a 128-byte physical memory. What base and bounds would you set up so as to get the simulator to generate the following translation results for the specified address stream: valid, valid, violation, ..., violation, valid, valid? Assume the following parameters: segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 ? --l0 ? --b1 ? --l1 ?

When base0 = 0, limit0=2, base1 = 15, limit1=2

python ./segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -b 0 -l 2 -B 15 -L 2 -c

4. Assume we want to generate a problem where roughly 90% of the randomly-generated virtual addresses are valid (not segmentation violations). How should you configure the simulator to do so? Which parameters are important to getting this outcome?
python .\segmentation.py -a 100 -p 128 -b 0 -l 45 -B 99 -L 45 -n 10 -c

When virtual address space is 100, each register limits are set at 50 * 90%, then almost 90% generated address are valid.

5. Can you run the simulator such that no virtual addresses are valid? How?
Set the limit0 and limit1 to 0
python .\segmentation.py -a 128 -p 512 -b 0 -l 0 -B 512 -L 0 -s 0 -c

Chapter 17

1. **First run with the flags -n 10 -H 0 -p BEST -s 0 to generate a few random allocations and frees. Can you predict what alloc()/free() will return? Can you guess the state of the free list after each request? What do you notice about the free list over time?**

Without merging the adjacent free memory, after a few alloc and free overtime, the free list is chopped into small chunks of fragments.

2. How are the results different when using a WORST fit policy to search the free list (-p WORST)? What changes?
The new alloc memory will find the biggest chunk of free memory to use. But this policy seems

to generate more fragments in free list than BEST, because without merging the adjacent free memory, the freed memory will just be returned as a fragment.

2. What about when using FIRST fit (-p FIRST)? What speeds up when you use first fit?
With first fit, the alloc memory finds the first big enough chunk of free memory to use. The search time speeds up because no exhaustive search is applied.

4. For the above questions, how the list is kept ordered can affect the time it takes to find a free location for some of the policies. Use the different free list orderings (-l ADDRSORT, -l SIZESORT+, -l SIZESORT-) to see how the policies and the list orderings interact.

First fit:
-l ADDRSORT free list is sorted by address, this seems to require medium numbers of searches, because the size of each chunk is random
-l SIZESORT+: free list is sorted from smaller size to bigger size, this requires the most numbers of searches
-l SIZESORT-: free list is sorted from bigger size to smaller size, this requires the least numbers of searches

5. Coalescing of a free list can be quite important. Increase the number of random allocations (say to -n 1000). What happens to larger allocation requests over time? Run with and without coalescing (i.e., without and with the -C flag). What differences in outcome do you see? How big is the free list over time in each case? Does the ordering of the list matter in this case?

Larger alloc can fail without -C, because of too many fragments and no big enough spaces.
Differences are mainly in the free list size, and the successful rate in larger allocs.
Size of free list: with -C usually less than 5, without C can be around 30
The ordering of the list matters, ADDRSORT can make merging easier.
6. What happens when you change the percent allocated fraction -P to higher than 50? What happens to allocations as it nears 100? What about as the percent nears 0?

When -P is higher than 50, this is similar to real world scenarios. More fragments in free list are created.
When allocations -P nears 100 or nears 0, the size of free list doesn't change. Because it either alloc from the fist free node in order, or alloc and immediately free the memory.

7. What kind of specific requests can you make to generate a highly fragmented free space? Use the -A flag to create fragmented free lists, and see how different policies and options change the organization of the free list.
Generate 10 malloc requests, then free the 0^{th}, 2^{nd}, 4^{th} requests.
python ./malloc.py -n 10 -H 0 -p BEST -A +5,+10,+15,+20,+25,+30,+35,+40,+45,+50,-0,-2,-4 -c