CS314 - Laboratory 7

Please note:

Deadline : 10th March 2024 (11.59 PM)

1. There are 4 main questions and each question has subquestions.
2. Answer each question in the order it appears and submit the final answers as rollno_lab6.pdf in the moodle.
3. You should be able to justify your answers and assumptions during the viva.
4. **For this lab, viva has more weightage. Without submission ,viva will not be conducted. [late submission will not be considered]**
5. **Plagiarized answers will fetch zero marks for this lab and also an appropriate disciplinary action.**

Q1. The program **relocation.py** allows you to see how address translations are performed in a system with base and bounds registers. See the README_base_bound for more details.

Questions

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.
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?
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?
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.

Q2. The program **segmentation.py** allows you to see how address translations are performed in a system with segmentation. See the README_segmentation for more details.

Questions

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
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512
  -L 20 -s 1
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512
  -L 20 -s 2
```

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?

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 ?
```

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?

5. Can you run the simulator such that no virtual addresses are valid? How?

Q3.The program **paging-linear-size.py** lets you figure out the size of a linear page table given a variety of input parameters.

Compute how big a linear page table is with the characteristics such as different number of bits in the address space, different page size, different page table entry size. Explain your answers for various cases.

Q4. You will use the program, "**paging-linear-translate.py"** to see if you understand how simple virtual-to-physical address translation works with linear page tables. See the README_paging for more details.

1. Before doing any translations, let's use the simulator to study how linear page tables change size given different parameters. Compute the size of linear page tables as different parameters change. Some suggested inputs are below; by using the `-v flag`, you can see how many page-table entries are filled. First, to understand how linear page table size changes as the address space grows, run with these flags:

```
-P 1k -a 1m -p 512m -v -n 0
-P 1k -a 2m -p 512m -v -n 0
-P 1k -a 4m -p 512m -v -n 0
```

   Then, to understand how linear page table size changes as page size grows:

```
-P 1k -a 1m -p 512m -v -n 0
-P 2k -a 1m -p 512m -v -n 0
-P 4k -a 1m -p 512m -v -n 0
```

   Before running any of these, try to think about the expected trends. How should page-table size change as the address space grows? As the page size grows? Why not use big pages in general?

2. Now let's do some translations. Start with some small examples, and change the number of pages that are allocated to the address space with the `-u flag`. For example:

```
-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
```

   What happens as you increase the percentage of pages that are allocated in each address space?

3. Now let's try some different random seeds, and some different (and sometimes quite crazy) address-space parameters, for variety:

```
-P 8  -a 32   -p 1024 -v -s 1
-P 8k -a 32k  -p 1m   -v -s 2
-P 1m -a 256m -p 512m -v -s 3
```

   Which of these parameter combinations are unrealistic? Why?

4. Use the program to try out some other problems. Can you find the limits of where the program doesn't work anymore? For example, what happens if the address-space size is *bigger* than physical memory?