Ch18

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
The page-table size is 1024.
-P 1k -a 2m -p 512m -v -n 0
The page-table size is 2048.
-P 1k -a 4m -p 512m -v -n 0
The page-table size is 4096.
Then, to understand how linear page table size changes as page size grows:
-P 1k -a 1m -p 512m -v -n 0
The page-table size is 1024.
-P 2k -a 1m -p 512m -v -n 0
The page-table size is 512.
-P 4k -a 1m -p 512m -v -n 0
The page-table size is 256.

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?

As the address space grows, the page-table size grows.
As the page size grows, the page-table size narrows down.
The size of a linear page table is size = address_space / page_size.
If the pages are very big, waste would occur.

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?

When I increased the percentage of pages that are allocated in each address space, there are more valid pages.

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

Virtual Address Trace
  VA 0x0000000e (decimal: 14) --> 0000030e (decimal 782) [VPN 1]
  VA 0x00000014 (decimal: 20) --> Invalid (VPN 2 not valid)
  VA 0x00000019 (decimal: 25) --> Invalid (VPN 3 not valid)
  VA 0x00000003 (decimal: 3) --> Invalid (VPN 0 not valid)
  VA 0x00000000 (decimal: 0) --> Invalid (VPN 0 not valid)

-P 8k -a 32k -p 1m -v -s 2

Virtual Address Trace
  VA 0x000055b9 (decimal: 21945) --> Invalid (VPN 2 not valid)
  VA 0x00002771 (decimal: 10097) --> Invalid (VPN 1 not valid)
  VA 0x00004d8f (decimal: 19855) --> Invalid (VPN 2 not valid)
  VA 0x00004dab (decimal: 19883) --> Invalid (VPN 2 not valid)
  VA 0x00004a64 (decimal: 19044) --> Invalid (VPN 2 not valid)

-P 1m -a 256m -p 512m -v -s 3

Virtual Address Trace
  VA 0x0308b24d (decimal: 50901581) --> 1f68b24d (decimal 526955085) [VPN 48]
  VA 0x042351e6 (decimal: 69423590) --> Invalid (VPN 66 not valid)
  VA 0x02feb67b (decimal: 50247291) --> 0a9eb67b (decimal 178173563) [VPN 47]
  VA 0x0b46977d (decimal: 189175677) --> Invalid (VPN 180 not valid)
  VA 0x0dbcceb4 (decimal: 230477492) --> 1f2cceb4 (decimal 523030196) [VPN 219]

Which of these parameter combinations are unrealistic? Why?

The first one and the second one are too small, the page-table sizes are only 4 for each.
For the third one, the page size is too large.

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 physicalmemory?

 1e18@Zuojuns-MacBook-Air vm-paging % ./paging-linear-translate.py -a 64k -v -c
ARG seed 0
ARG address space size 64k
ARG phys mem size 64k
ARG page size 4k
ARG verbose True
ARG addresses -1

Error: physical memory size must be GREATER than address space size (for this simulation)

1e18@Zuojuns-MacBook-Air vm-paging % ./paging-linear-translate.py -a 0 -v -c
ARG seed 0
ARG address space size 0
ARG phys mem size 64k
ARG page size 4k
ARG verbose True
ARG addresses -1

Error: must specify a non-zero address-space size.

1e18@Zuojuns-MacBook-Air vm-paging % ./paging-linear-translate.py -p 0 -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 0
ARG page size 4k
ARG verbose True
ARG addresses -1

Error: must specify a non-zero physical memory size.

1e18@Zuojuns-MacBook-Air vm-paging % ./paging-linear-translate.py -P 32k -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 64k
ARG page size 32k
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)

Virtual Address Trace
Traceback (most recent call last):

```
  File "./paging-linear-translate.py", line 175, in <module>
    if pt[vpn] < 0:
IndexError: array index out of range


1e18@Zuojuns-MacBook-Air vm-paging % ./paging-linear-translate.py -P 0 -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 64k
ARG page size 0
ARG verbose True
ARG addresses -1

Traceback (most recent call last):
  File "./paging-linear-translate.py", line 86, in <module>
    mustbemultipleof(asize, pagesize, 'address space must be a multiple of the pagesize')
  File "./paging-linear-translate.py", line 15, in mustbemultipleof
    if (int(float(bignum)/float(num)) != (int(bignum) / int(num))):
ZeroDivisionError: float division by zero
```

Ch19

1. For timing, you'll need to use a timer (e.g., gettimeofday()). How precise is such a timer? How long does an operation have to take in order for you to time it precisely? (this will help determine how many times, in a loop, you'll have to repeat a page access in order to time it successfully)

clock_gettime(CLOCK_PROCESS_CPUTIME_ID)
Precision is 1 nano sec.
As long as the operation would take more than 1 nano second, we can measure it successfully. Or we can operate it for more than 1 time, and then calculate it by dividing the total time consumption with running times.

2. Write the program, called tlb.c, that can roughly measure the cost of accessing each page. Inputs to the program should be: **the number of pages to touch** and **the number of trials**.

The program is attached in tlb.c

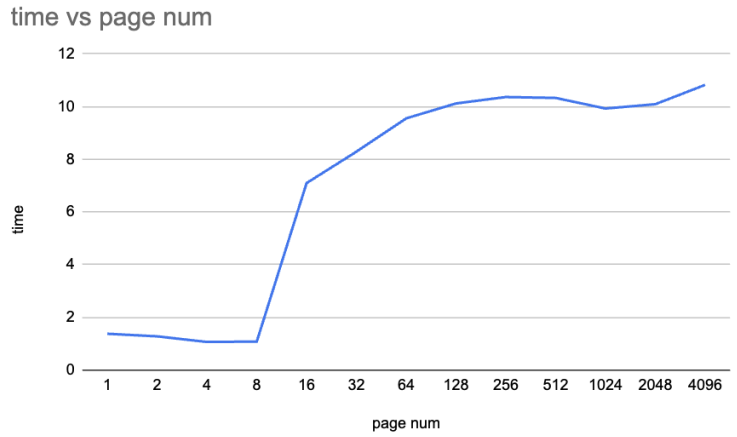
3. Now write a script in your favorite scripting language (bash?) to run this program, while varying the number of pages accessed from 1 up to a few thousand, perhaps incrementing by a factor of two per iteration. Run the script on different machines and gather some data. How many trials are needed to get reliable measurements?

```
(base) zhuzuojun@wsg5:~/learn/ostep/ch19$ python run.py
page num 1
```

1.488900 nano seconds
page num 2
1.280800 nano seconds
page num 4
1.068200 nano seconds
page num 8
1.082888 nano seconds
page num 16
7.096287 nano seconds
page num 32
8.288213 nano seconds
page num 64
9.553108 nano seconds
page num 128
10.117922 nano seconds
page num 256
10.362133 nano seconds
page num 512
10.327808 nano seconds
page num 1024
9.926177 nano seconds
page num 2048
10.089313 nano seconds
page num 4096
10.821908 nano seconds

When the number of trials reached to 10000, the measurement reached the second-level TLB and the result becomes reliable.

4. Next, graph the results, making a graph that looks similar to the one above. Use a good tool like ploticus or even zplot. Visualization usually makes the data much easier to digest; why do you think that is?



The picture should be in the same pattern with that one of the question.
5. One thing to watch out for is **compiler optimization**. Compilers do all sorts of clever things, including removing loops which increment values that no other part of the program subsequently uses. How can you ensure the compiler does not remove the main loop above from your TLB size estimator?
Add -O0 when compiling.
gcc -o0 tlb.c -o tlb

6. Another thing to watch out for is the fact that most systems today ship with multiple CPUs, and each CPU, of course, has its own TLB hierarchy. To really get good measurements, you have to run your code **on just one CPU,** instead of letting the scheduler bounce it from one CPU to the next. How can you do that? (hint: look up “pinning a thread” on Google for some clues) What will happen if you don’t do this, and the code moves from one CPU to the other?
Use taskset command
taskset -c 0 ./tlb param1 param2

7. Another issue that might arise relates to initialization. If you don’t initialize the array a above before accessing it, the first time you access it will be very expensive, due to initial access costs such as demand zeroing. Will this affect your code and its timing? What can you do to counterbalance these potential costs?
Yes, it will affect the timing.
I solved this problem by initializing the array before timing.