

CSC3150 Assignment3 Report

Ziqi Gao 高梓骐

118010077

1. Introduction

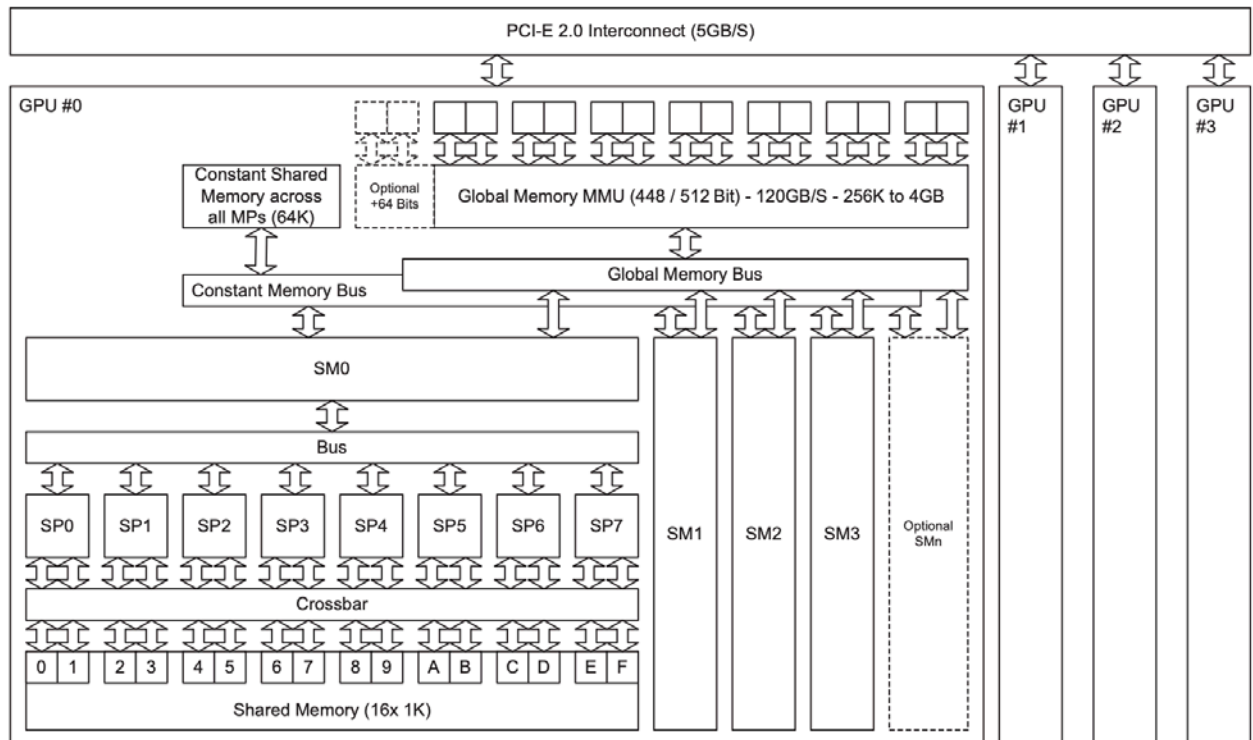
Versions of OS: Windows 10 Pro - 1909

Compilation Toolchains:

C++ Compiler: MSVC-14.27.29110 x64 architecture.

CUDA Compiler: NVCC 11

In this task, I will simulate a single process virtual memory system with CUDA. We will use one GPU core to execute the simulation. The global memory of the core is larger and slower, so it can be considered as the secondary memory in the virtual memory system. Moreover, the shared memory that is smaller and faster will be considered as our main memory.



In this program, we will simulate the process of writing data to the memory and the process of reading data from memory. We will make a page table to map the logical address and the physical address. Whenever we try to get access to a location not in

our main memory, we will raise the page fault and count the number of the page fault number happens during the process. The result of the reading process will be written into a file called *snapshot.bin*.

2. Basic Design

This program consists of two parts. The first part is executed in our CPU, RAM, and disk. Another CUDA part is executed in the GPU. Those functions and variables that will be executed in GPU will have name identifiers like `__global__`, or `__device__`.

In the first part, we will only declare some variables and one main function that will pass those arguments to the GPU. After the GPU finishes the simulation, CPU reads the data from GPU's buffer and writes those data to the *snapshot.bin*.

Those functions and variables are defined in *main.cu*.

Also, there are some functions and variables declared to be used in the GPU in *main.cu*. mykernel function will start the simulation by calling the `use_program` function in *user_program.cu*.

This `user_program` function will load data from the data buffer to the simulated main memory. Then it will look through the entire pages in the memory. Finally, it will call `vm_snapshot` function to write all the data got from the input buffer in the main memory and the secondary memory to the data buffer.

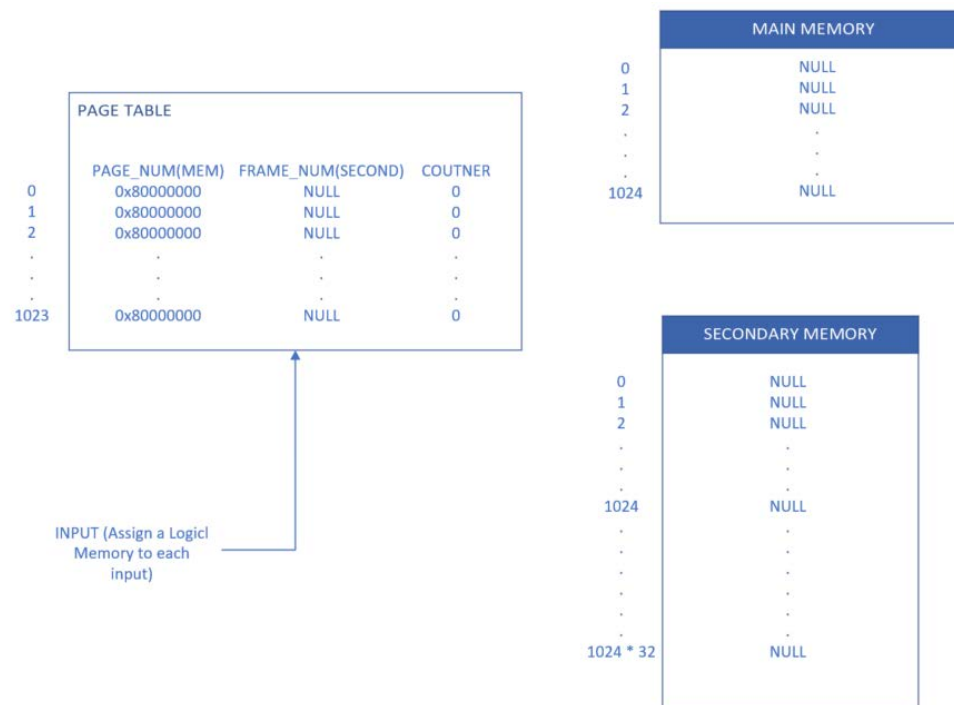
As to the virtual memory system, here are some figures showing how it works: The simulation process can be divided into the following parts:

a. Insert the first 32KB data:

At the beginning, the page table's first column (index: 0-1023) is set to 0x80000000 indicating that 0-1023's memory space is empty. When data comes in, it will raise a page fault since there is no corresponding physical memory for the data's logical memory (the logical memory address is equal to the order of the input). In practical, we will keep a free-frame list, but in this simulation, I do not work in this way. The page table works like a hash table, whenever the logical address comes in, it will first check the location calculated by the following hash function:

$$\text{logical address} \bmod 1024 \text{ (Length of the Table)}$$

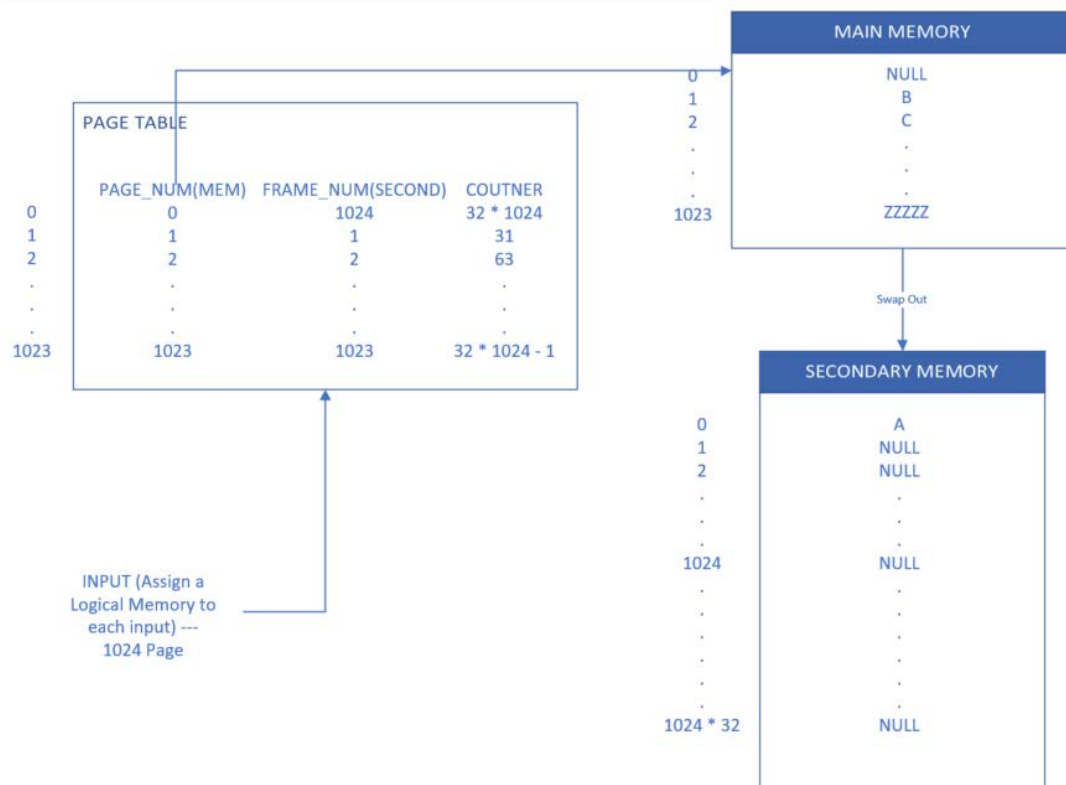
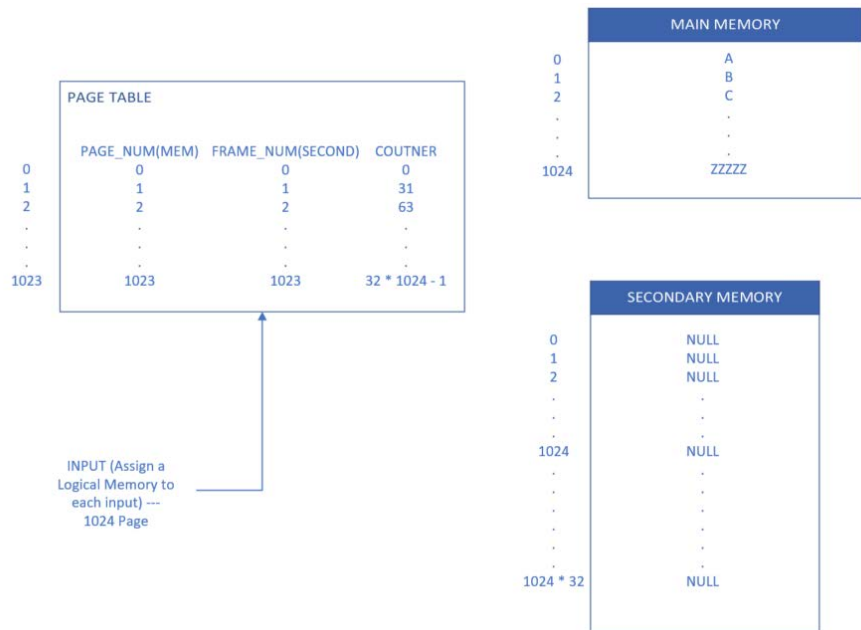
If the memory's number is 0x80000000, it raises the page fault, and the operating system will conduct the LRU (least recently used) algorithm to allocate a new physical memory number to it. In this stage, all the allocated new physical memory number should be equal to the remainder, i.e., the result of the hash function. Also, the frame number in the secondary memory is also allocated according to the page number of the logical memory.



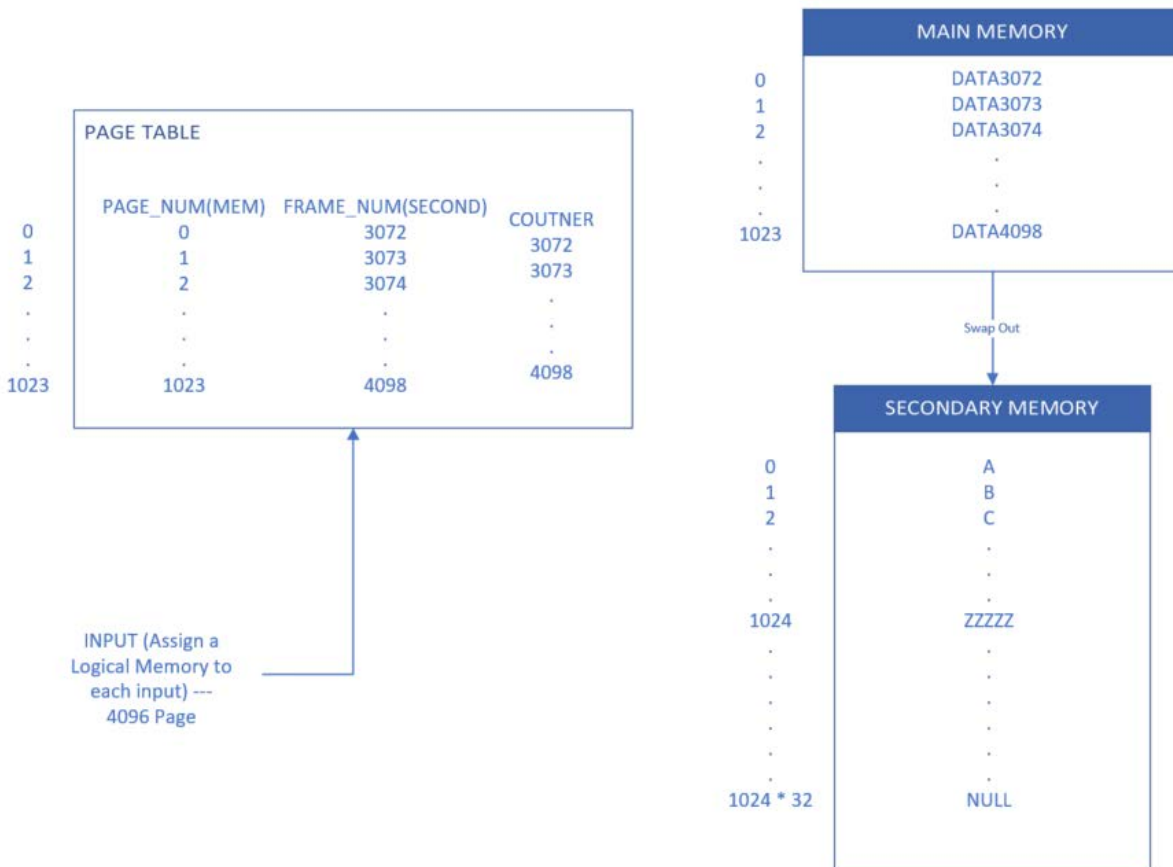
b. Insert the 32KB – 128KB Data:

When we are trying to insert more data into the already full memory, we will need to use another strategy. For example, when we are inserting the first byte of the 1024 page. First of all, the hash table will map the logical address to the 0th row of the page table. And it will find that the FRAME_NUM is not equal to the page number of that input value. Then we will conduct the LRU algorithm and find that the 0th row has the smallest counter value. Then we will do the following steps:

- Raise the page fault.
- Swap out the data in the main memory and save it to the 0th row of the secondary memory according to the FRAME_NUM in the page table.
- Update the page table with the three values to be 0, 1024, 1024 * 32.

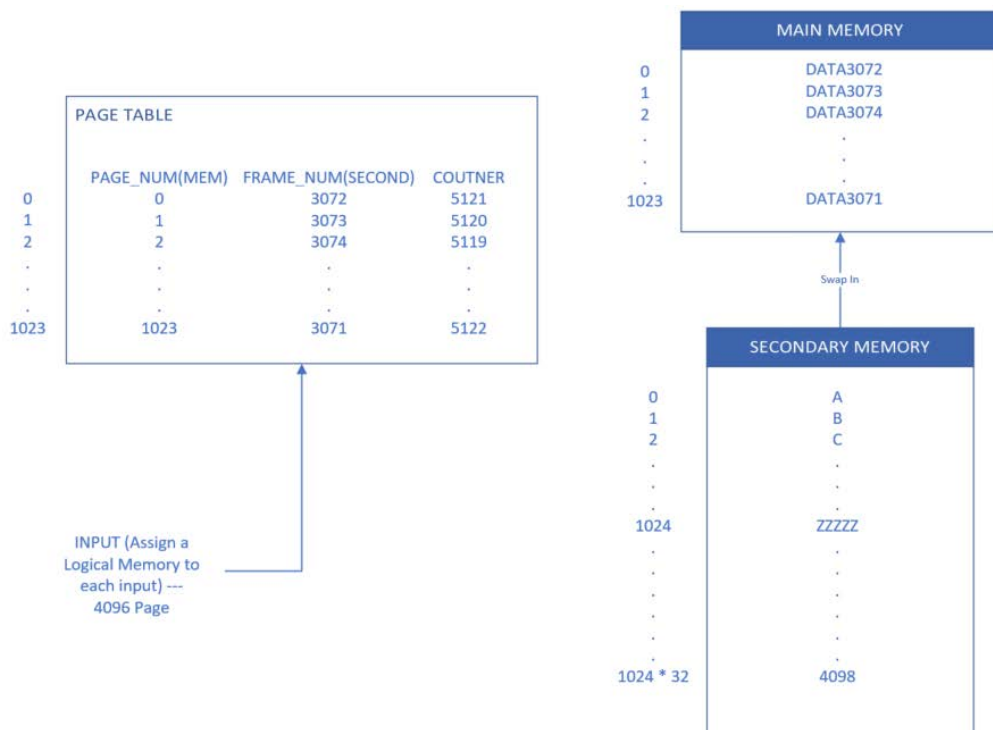


After all the input are finished, those three tables look like this:



c. Read 32769 Times in the Reverse Order:

In this step, we will read the data in reverse order. $32769 = 1024 \text{ Page} + 1 \text{ Byte}$. As a result, this step will go through all of the data in the main memory and one extra bit from the secondary memory by swapping in. The memory after the first 32768 times read will become:



d. Read from the 0 page (Snapshot):

This step will use `vm_read` function for many times to read data from the memory and the secondary memory. The process is almost the same as the c part.

3. LRU Algorithm and Data Structure:

The structure of my page table is a one-dimension array. However, it has $4 * 1024 = 4096$ rows = 16KB. I consider the 0-1023 row as the `PAGE_NUM`, 1024-2047 row as the `FRAME_NUM`, 2048-3071 rows as the `COUNTER`, and 3072-4095 rows as the modify bit.

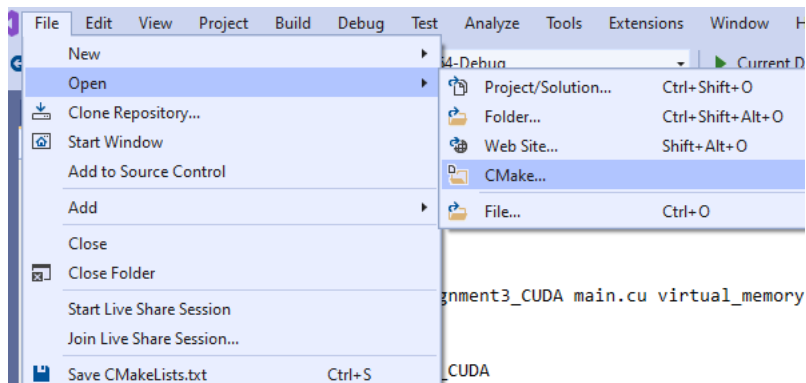
I conduct the LRU algorithm by adding a counter to the page table. Whenever I update one row in the page table, memory, or secondary memory, the corresponding row's counter will plus 1. The row with its least value of the counter will be replaced if we need to execute the swapping.

4. Execution

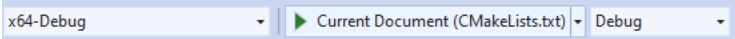
I did not develop my project in VS because I am not familiar with its compiler configuration, and I met MSB3721 fault and I cannot figure it out. Therefore, I will also upload my CLion project to you, if my VS project cannot be built, please use it. Please use CLion or other IDE to open the cmake file to build my project, thanks!

For Visual Studio Users:

File >> Open >> CMake >> Open my CMakeLists.txt



After you import the cmake file, please wait for a moment and click the Current Document (CMakeLists.txt) to run my code. (Please copy the data.bin to the debug file, otherwise the program cannot find this file.)

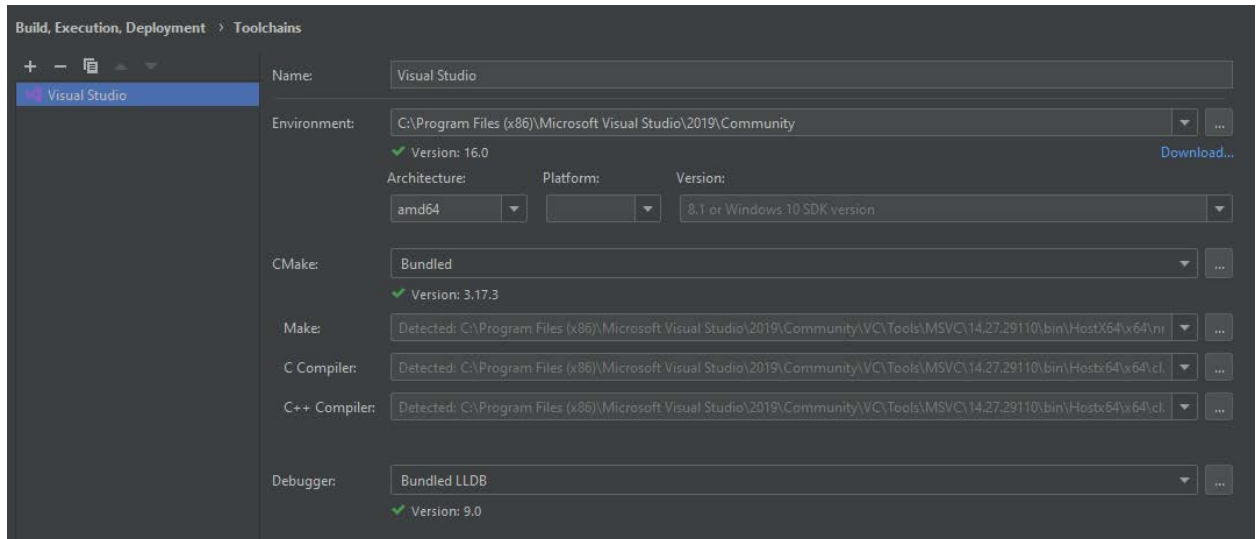


CLion Users:

Click open a new project and choose my CMakeLists.txt to load my project.

After you import the cmake file, please wait for a moment run my file.

Since I am using CLion as my IDE to code this program, I have a CMAKE_FILE, and the setting of the CLion Toolchains is as follows:



What I am doing during my development is to save the data.bin with my source code, and run them in IDE. Then I can check my snapshot.bin in the inner cmake-build-debug directory.

Also, I changed the virtual memory.h to virtual memory.cuh to link my code. I come across link error if I use virtual memory.h.

The output of my program is as follows:

At the beginning, it will output the input size of the file. Whenever it read a byte, it will output the byte's index in the memory.

```
input size: 131072
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
INDEX: 1023
```

In the end, it prints out the total page fault number.


```
INDEX: 340
INDEX: 340
INDEX: 340
INDEX: 340
INDEX: 340
pagefault number is 8193

Process finished with exit code 0
```

$8193 = 4196 + 1 + 4196$, the first 4196 comes from the write part. Whenever we write a page to the memory, it will raise a page fault.

The 1 comes from the 32769 times read. Since $32769 = 1024 \text{ pages} + 1 \text{ byte}$. The last 1 byte's page number is not in the memory so it will raise the page fault.

The second 4196 comes from the snapshot part. Since we read it from the first byte to the last byte, we do not have any needed content in the beginning. As a result, every page read from the memory will raise a page fault.

My program will execute for about 20 seconds, which is a not bad speed.

And the snapshot.bin looks like that:

00000000	37	0C	60	51	38	5A	25	0B	13	2A	2A	01	0C	07	05	2F	7. `Q8Z%..**.... /
00000010	62	0B	3A	2B	3E	35	2B	3C	0E	27	60	3B	04	53	2A	3A	b.:+>5+<.'`.:S*:
00000020	5E	5A	5B	31	1F	1B	0B	01	14	04	01	20	0B	05	1E	08	`Z[1..... ..
00000030	44	57	02	1D	5C	61	28	05	23	58	3F	5A	46	39	63	3F	DW..\a(.#X?ZF9c?
00000040	62	59	3F	1C	44	49	1C	57	1C	51	46	5A	55	63	32	34	bY?.DI.W.QFZUc24
00000050	56	33	21	1D	2F	18	21	21	0B	30	16	50	38	15	2A	35	V3!./.!..0.P8.*5
00000060	3D	38	20	1C	50	3B	43	08	5B	24	31	4C	23	62	4F	48	=8 .P;C.[\$1L#bOH
00000070	01	3F	64	63	57	54	20	31	1F	05	1D	56	4D	16	5A	26	.?dcWT 1...VM.Z&
00000080	1E	15	11	09	20	53	44	16	13	11	31	05	0E	1C	4C	0E SD...1...L.
00000090	2A	1B	41	50	0A	60	1D	5D	34	39	1E	51	1E	14	46	3B	*.AP.`.]49.Q..F:
000000a0	5C	56	44	4B	45	57	61	27	03	61	2B	45	4C	46	22	12	\VDKEWa'.a+ELF".
000000b0	60	62	61	39	2D	19	31	61	21	4F	4D	3F	32	62	49	29	`ba9-.1a!OM?2bI)
000000c0	53	5C	10	03	4F	40	5D	51	3C	57	31	24	38	23	05	03	S\..0@]Q<W1\$8#..
000000d0	20	01	0C	4D	4E	3C	19	0A	5A	35	18	5B	32	31	20	54	..MN<..Z5.[21 T
000000e0	28	63	57	12	0E	1F	33	49	12	33	3C	49	55	40	1C	45	(cW...3I.3<IU@.E
000000f0	11	27	2D	2E	32	15	37	28	49	1F	1E	4A	4F	0D	39	46	.'-.2.7(I..JO.9F
00000100	3F	5F	58	4C	1A	5A	01	2B	28	3C	43	4D	4C	5E	2D	2C	?_XL.Z.+(<CML^-,
00000110	20	29	59	22	3D	5F	19	55	4D	36	3A	37	13	42	19	51)Y"=..UM6:7.B.Q
00000120	3D	40	09	26	35	09	50	2C	14	2E	14	5F	5C	40	5A	4B	=@.&5.P,...\@ZK
00000130	38	4E	08	10	19	20	34	01	26	09	38	08	1B	20	28	57	8N... 4.&.8.. (W
00000140	2F	30	4C	33	38	37	5E	4C	34	42	16	5F	51	0C	46	25	/OL387`L4B._Q.F%
00000150	29	1D	34	41	3D	04	42	32	40	49	39	2A	38	60	50	02).4A=.B2@I9*8`P.
00000160	2C	37	34	33	3D	61	1A	0D	3E	64	3B	2B	0B	1C	4F	03	,743=a..>d;+..0.
00000170	39	52	44	45	25	21	46	35	39	4E	5E	0C	49	4A	0D	44	9RDE%!F59N`.IJ.D
00000180	1C	10	47	59	0C	30	35	1A	2F	0B	14	09	5B	32	0C	63	..GY.05./... [2.c
00000190	1F	1F	13	14	3F	58	48	47	41	11	52	59	2A	5E	09	16?XHGA.RY*`..
000001a0	3D	4F	3E	48	1A	0E	31	19	4C	44	21	12	45	60	10	34	=O>H..1.LD!.E`.4
000001b0	1A	22	47	28	49	5E	0A	25	3E	5B	1A	04	24	22	19	60	."G(I`.%>[.\$.`.
000001c0	40	56	14	59	33	44	0D	1A	58	62	60	38	5D	0B	07	47	@V.Y3D..Xb`8]..G
000001d0	61	1D	0A	15	16	14	3A	54	3E	23	27	62	14	3F	5D	53	a.....:T>#`b.?]S
000001e0	30	0C	17	62	20	58	4B	13	25	16	1A	1D	55	55	33	51	0..b XK.%...UU3Q
000001f0	0D	3D	01	57	20	0A	16	5D	60	3C	5A	0F	4A	23	31	15	.=.W ..]`<Z.J#1.
00000200	62	18	46	1D	0B	60	63	63	12	19	1B	36	3D	4E	22	49	b.F..`cc...6=N"I
00000210	5A	56	0B	15	30	20	41	2B	2B	07	0A	10	29	3A	58	26	ZV..0 A++...):X&
00000220	21	09	13	2B	05	11	5D	4A	5D	14	1B	35	31	0C	4E	5A	!...+...]J]..51.NZ
00000230	31	58	0A	30	48	1A	5B	0E	20	34	52	48	3D	45	3E	5E	1X.OH.[. 4RH=E>^
00000240	4E	20	58	22	64	51	3B	5D	34	25	61	64	64	1A	29	30	N X"dQ:]4%add.)0
00000250	42	02	60	59	1B	26	36	3B	59	23	52	31	38	5F	5E	21	B.`Y.&6;Y#R18_`!
00000260	1A	52	12	1A	3E	1C	46	0D	40	12	40	3F	2C	04	3E	3D	.R...>.F.@?.,.>=
00000270	39	09	31	53	2E	02	29	56	25	4B	57	5C	45	50	4C	5F	9.1S..)V%KW\EPL_
00000280	3D	2D	14	4A	48	29	56	23	3A	01	31	01	38	0A	3D	0C	=-.JH)V#:.1.8.=.
00000290	13	3D	5F	10	3F	57	02	33	3D	28	5E	52	47	45	4C	20	..=?W.3=(^RGEL
000002a0	41	2F	39	24	57	2B	16	2C	5F	16	61	33	1F	09	3E	01	A/9\$W+.,_.a3...>.
000002b0	46	38	11	54	5F	46	22	37	3D	1B	24	1F	2F	3F	0E	0B	F8.T_F"7=.\$./?..
000002c0	09	47	62	2F	0D	47	5B	07	5C	27	09	4A	2F	47	1B	10	.Gb/.G[.\`.J/G..
000002d0	1A	2D	62	40	40	5A	1B	4C	2E	0E	06	00	1B	1A	46	5D	..UMAT I` .. EV

5. What I Learned

I learned lots of knowledge about the Virtual Memory Schema for one process memory management through this project. It is really hard to work out how this schema works because it is abstract and tedious. I need to draw lots of figures to

help me understand the process. Also, the writing process is much different from the content introduced in the textbook that only covers the read part.

There is still something I need to get improved. For example, I did not use a common data structure like linked list or stack. I use a one-dimension array to represent my page table. I think the speed is fine for this data structure, but I think stack and linked list may be more space efficient because those two structures' own properties can show the order of the data without the counter.