CSCE 313 QUIZ 2 Spring 2021

Name: Arthur UIN 327003368
Total Points: -------/100

1. [5 pts] While implementing the process state diagram, what is the problem of having only 1 queue for all blocked processes waiting for all events? What is the solution to this problem? Describe with an example.

   Having only one queue for all blocked processes certainly has a problem because not all the blocked processes are requesting the same resource. For example, some process might be waiting for user input while another process is waiting for response of disk for a file request. Since the waiting time for each type of event varies a lot, having them all in one queue is not appropriate. The solution is this problem is to have multiple waiting queue for each of the event. The processor decides the current process should be waiting in which of the blocked queue, which is also known as multiple blocked queues. For example, the first queue can be used to store processes being kicked out for reaching end of time share, the second queue can store processes those are waiting for disk access, and the third queue can be used to waiting for user input.

2. [25 pts] Assume the following processes A, B, C are loaded in memory of a system that uses both multiprogramming (MP) and timesharing (TS) techniques. These processes have 15, 7, and 13 instructions respectively. Also assume that the dispatcher lives at address 100 in memory and spans 4 instructions (i.e., 100-103). The time quantum is long enough for exactly **5 instructions.**

   The following table shows only instruction addresses in the memory with I/O requests labelled, along with the duration of these I/O operations in terms of CPU instructions. Although I/O operations do not take CPU instructions, the duration means that the I/O operations will finish by the time the corresponding number of CPU instructions execute.

   Please draw one possible trace of these 3 processes running together in the CPU. Use page 14-15 of Lecure 3 for reference. You may skip the first invocation of the dispatcher to decide the first process to run in the CPU.

Process A	Process B	Process C
5000 5001 5002 5003 5004 (I/O, takes **6 ins**.) 5005 (I/O, takes **4 ins**.) 5006	8000 8001 8002 8003 (I/O, takes **7 ins.**) 8004 8005 8006	12000 12001(I/O, takes 3 ins.) 12002 12003 12004 12005 12006(I/O, takes 1 ins.)

5007 5008(I/O, takes **5 ins.**) 5009 5010 5011 5012 5013 5014		12007 12008(I/O, takes 2 ins.) 12009 12010 12011 12012

	Timeout	I/O request	End of process
"index" of instance	5	5	31
	62	13	70
		19	75
		24	
		40	
		47	
		53	

1	5000	26	101			
2	5001	27	102	51	103	
3	5002	28	103	52	12007	
4	5003	29	8004	53	12008	
5	5004	30	8005	54	100	
6	100	31	8006	55	101	
7	101	32	100	56	102	
8	102	33	101	57	103	
9	103	34	102	58	5009	
10	8000	35	103	59	5010	
11	8001	36	12002	60	5011	
12	8002	37	12003	61	5012	
13	8003	38	12004	62	5013	
14	100	39	12005	63	100	
15	101	40	12006	64	101	
16	102	41	100	65	102	
17	103	42	101	66	103	
18	12000	43	102	67	12009	
19	12001	44	103	68	12010	

20	100	45	5006	69	12011
21	101	46	5007	70	12012
22	102	47	5008	71	100
23	103	48	100	72	101
24	5005	49	101	73	102
25	100	50	102	74	103
				75	5014

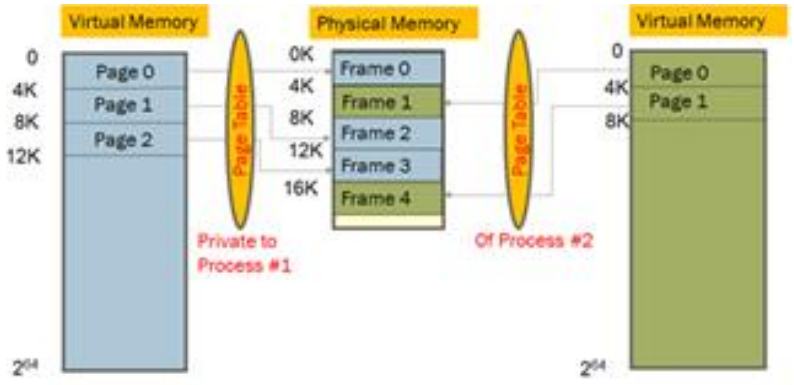
3. [5 pts] What is the difference between the "New" state and the "Ready to Run" state in the process state diagram?

   A process in new state indicates that the process is used in memory management. However, this is different from ready to run because ready to run requires both process's PCB and executable image in memory and there is space available in main memory for the new process to be loaded.

4. [5 pts] Is a transition from the "Blocked" state to directly to the "Exit" state possible in the process state diagram? How?

   This is totally possible, for example, a blocked child process is killed by its parent process thus the child goes from blocked to exit.

5. [10 pts] Assume that the following physical memory is full of already allocated 5 pages as shown below (i.e., it is 20KB in capacity). Describe what happens if process 2 wants to allocate and use another page. What changes in the page tables and the physical memory?



   When the physical memory is already full but one process tries to allocate another page, the least recently used frame will be replaced with the newly allocated frame then be placed at the disk. Additionally, the page table will mark the replaced frame is placed at the disk. So when the owner of the original frame tries to call it back, system will check if the physical memory has available space.

6. [25 pts] Draw the Descriptor Table (DT) and File Table (FT) for each possible process just after executing the lines 2, 5 and 10 and explain what you see in the standard output and **file.txt** along the way. FT entries should contain the `cursor` and the `reference count`

```
1.int fd=open("file.txt", O_CREAT|O_RDWR);
2.write (fd, "Hello world", 6);
3.if (!fork()){
4.  dup2 (fd, 1); //redirect stdout
5.  cout << "Mars, I am here!!" << endl;
6.}else{
7.  wait(0);
8.  lseek (fd, 0, SEEK_SET);
9.  cout << "Mars is great" << endl;
10. write (fd, "Hola  ", 5); //draw tables
11. close (fd);
12.}
```

Line 1 opens the file thus the file table has file.txt at the first row.
At line 2, file.txt is written with 6 chars, which will be "Hello ".

DT FT

Ref_cnt

0	Stdin	1	file.txt
1	Stdout		
2	stderr		
3			

Since line 4 alters the output from stdout to fd, any cout text at line 5 will be write into the file that fd points to thus the content of the file is now "Hello Mars, I am here!!". Note that the output does not over write the previous content because they share the same file offset and two file pointers are pointing to the same file.

DT FT

Ref_cnt

0	Stdin	2	file.txt
1	fd		
2	stderr		
3			

At line 10, since the process was forked, this process uses a new file table thus a new file offset, Therefore, the content of the file is now "Hola ".

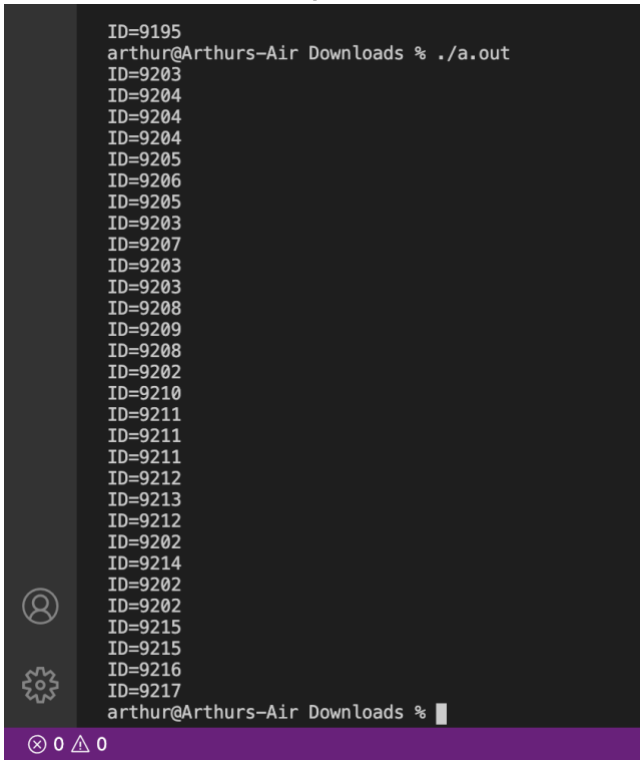
DT FT

		Ref_cnt	
0	Stdin	3	file.txt
1	fd		
2	stderr		
3			

7. [25 pts] Please explain the output of the following program using a process tree diagram as shown in Lecture 4 page 23. Assuming the main process's ID is 1000, explain the order of process creation (in fact, the number itself should show what order the processes are created). Also explain what you see in the standard output. Are different outputs possible for this program? Why or why not?

```
for (int i=0; i<4; i++){
    int cid = fork ();
    if (i < 2)
        wait (0);
    cout << "ID=" << getpid () << endl;
}
```

Sample output of the code, note that the main process is not 1000 in this sample.



Different outputs are possible for this code, because as the processes are running concurrently, the order of cout cannot be ensured, thus the output differs.

The process tree and output tree for each iteration: