CS 453/552: Operating Systems

Final Examination (Spring 2020)

Time: 120 minutes *Name :*______________________ *Total Points: 200*

- *This exam has 8 questions, for a maximum of 200 points.*

Question	Points	Score
1	20	
2	30	
3	20	
4	30	
5	20	
6	30	
7	30	
8	20	
Total:	200	

1. (20 points) **The Forked Nursery**. Examine the following C program fragment. How many **new** processes are created by the following code?

```
/* process A */
    /* ... */
    fork();
    fork();
    fork();
    fork();
    fork();
    fork();
    /* ... */
```

2. (30 points) **Draw the file system state**. A simulation trace is generated below using the tool vsfs.py. Assumption: when a file or directory is created, if allocating an new inode is needed, the file system will always try to allocate first available inode; or if allocating a new data block is needed, the file system will always try to allocate the first available data block.

```
Initial state (state 0)
inode bitmap  10000000
inodes        [d a:0 r:2] [] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0)] [] [] [] [] [] [] []

state 1

inode bitmap  11000000
inodes        [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []
data bitmap   11000000
data          [(.,0) (..,0) (xyz,1)] [(.,1) (..,0)] [] [] [] [] [] []

state 2

inode bitmap  11100000
inodes        [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [] [] [] [] []
data bitmap   11000000
data          [(.,0) (..,0) (xyz,1) (b,2)] [(.,1) (..,0)] [] [] [] [] [] []
```

```
state 3

inode bitmap  11110000
inodes        [d a:0 r:3] [d a:1 r:2] [f a:-1 r:1] [f a:-1 r:1] [] [] [] []
data bitmap   11000000
data          [(.,0) (..,0) (xyz,1) (b,2)] [(.,1) (..,0) (q,3)] [] [] [] [] [] []
```

1. Based on this trace, which operation has taken place in between state 0 and state 1? Hints: It is either a create or a mkdir operation, and create() is used to create an empty file, while mkdir() is used to create a directory. Choose one from these two options, and make sure you specify the parameter: the full path of the file or directory created. For example, if you believe a directory /a/b/c is just created, then say mkdir("/a/b/c"); and if you believe a file is /d/e is created, then say create("/d/e"). (10 points)

2. Based on this trace, which operation has taken place in between state 1 and state 2? Hints: again, it is either a create or a mkdir operation. Please specify the full path. (10 points)

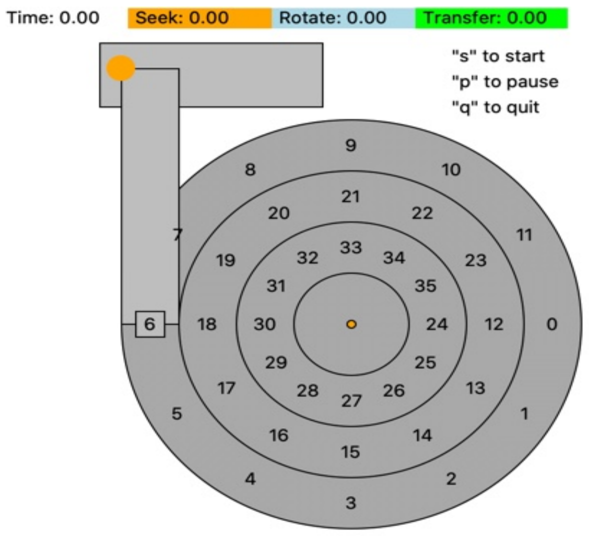
3. Based on this trace, which operation has taken place in between state 2 and state 3? Hints: again, it is either a create or a mkdir operation. Please specify the full path. (10 points)

3. (20 points) **Swaping: Paging Policy: LRU**. A simulation trace is generated below using the tool paging-policy.py. Assuming a replacement policy of LRU, and a cache of size 4 pages, figure out whether each of the following page references hit or miss in the page cache.

```
Access: 9  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 7  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 2  Hit/Miss?  State of Memory?
Access: 6  Hit/Miss?  State of Memory?
Access: 9  Hit/Miss?  State of Memory?
Access: 8  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
Access: 1  Hit/Miss?  State of Memory?
```

Note: Your answer needs to say MISS or HIT, as well as which pages are in the memory. No need to calculate the hit rate.

4. (30 points) **disk**. As shown this figure - generated by the simulator disk.py, the disk head's initial position is at sector 6. Compute the seek, rotation, and transfer times for the following request: sector 13. Assumptions: the disk platter rotates 1 degree per time unit, seeking from one track to the right next track takes 30 time units, transfering (i.e., reading from or writing to) a sector takes 30 time units. Transferring starts when the disk head is 15 degrees ahead of the target sector, ends when the disk head is 15 degrees after the target sector. Note that this disk spins **clockwise**.



5. (20 points) **The blocks we build when we first practice to go big.** The following fragment of code is from `ext4.h` in the Linux kernel source code.

```
#define EXT4_MIN_BLOCK_SIZE     1024
#define EXT4_MAX_BLOCK_SIZE     65536
/*
 * Constants relative to the data blocks
 */
#define EXT4_NDIR_BLOCKS        12 // 12 direct pointers
#define EXT4_IND_BLOCK          EXT4_NDIR_BLOCKS // 1 single indrect pointer
#define EXT4_DIND_BLOCK         (EXT4_IND_BLOCK + 1)
#define EXT4_TIND_BLOCK         (EXT4_DIND_BLOCK + 1)
#define EXT4_N_BLOCKS           (EXT4_TIND_BLOCK + 1)
/*
 * Structure of an inode on the disk
 */
struct ext4_inode {
    ...
    __le32   i_block[EXT4_N_BLOCKS];/* Pointers to blocks */
    ...
}
```

Assume a block size of **4096 bytes** and the addresses are **64-bit**, answer the following questions for the ext4 file system. Please express your answers in appropriate units like KB, MB, GB, TB, PB etc.

- What is the maximum size of a file using only direct pointers?

- What is the maximum size of a file using direct and single indirect pointers?

6. (30 points) **Address Translation: Paging**. A simulation trace is generated below using the tool paging-linear-translate.py.

```
ARG address space size 8k
ARG phys mem size 24k
ARG page size 512
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.
Page Table (from entry 0 down to the max size)
  [       0]   0x8000002d
  [       1]   0x80000004
  [       2]   0x80000023
  [       3]   0x8000000e
  [       4]   0x8000001d
  [       5]   0x80000007
  [       6]   0x80000012
  [       7]   0x8000002f
  [       8]   0x8000001a
  [       9]   0x8000000c
  [      10]   0x80000001
  [      11]   0x8000000f
  [      12]   0x8000002a
  [      13]   0x80000006
  [      14]   0x80000008
  [      15]   0x80000026

Virtual Address Trace
  VA 0x00001d01 (decimal:     7425) --> PA or invalid address?
  VA 0x00001945 (decimal:     6469) --> PA or invalid address?
  VA 0x00000b52 (decimal:     2898) --> PA or invalid address?
For each virtual address, write down the physical address it translates to
OR write down that it is invalid.
```

7. (30 points) **Address Translation: Paging**. A simulation trace is generated below using the tool paging-linear-translate.py.

```
ARG address space size 64
ARG phys mem size 512
ARG page size 8

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.

Page Table (from entry 0 down to the max size)
  [       0]   0x8000001b
  [       1]   0x8000000d
  [       2]   0x80000034
  [       3]   0x8000000a
  [       4]   0x80000014
  [       5]   0x00000000
  [       6]   0x8000003f
  [       7]   0x00000000

Virtual Address Trace
  VA 0x00000037 (decimal:       55) --> PA or invalid address?
  VA 0x00000018 (decimal:       24) --> PA or invalid address?
  VA 0x0000002b (decimal:       43) --> PA or invalid address?

For each virtual address, write down the physical address it translates to
OR write down that it is invalid.
```

8. (20 points) **Multiprocessor Scheduling**. Below is a workload which includes 3 jobs (job a,b,c), on a two-CPU system, the working set size of each job is 5. Please calcuate how much time it takes to finish the whole workload. Assume we are using a round robin scheduler.

```
ARG job_num 3
ARG num_cpus 2
ARG quantum 1
ARG cache_size 100

Job name:a run_time:8
Job name:b run_time:8
Job name:c run_time:2
```

Note 1: cache warm up time: 2 time units. i.e., if a job runs for 2 time units, the cache on that CPU becomes warm, and then the job starts running faster.
Note 2: cache warm up rate: 2x. i.e., a job will execute at a 2x speed when the cache becomes warm.
Note 3: below is an EXAMPLE trace, it shows the meaning of "a round robin scheduler" on a two-CPU system. If the whole workload finishes at the end of time unit 4, then just say the answer is "Finished time: 5".

```
======
Scheduler central queue: ['a', 'b', 'c']

   0   a [  3] cache[   ]     b [  2] cache[   ]
---------------------------------------------------
   1   c [  1] cache[   ]     a [  2] cache[   ]
---------------------------------------------------
   2   b [  1] cache[   ]     c [  0] cache[   ]
---------------------------------------------------
   3   a [  1] cache[w  ]     b [  0] cache[ w ]
---------------------------------------------------
   4   a [  0] cache[w  ]     - [   ] cache[ w ]

Finished time 5 (This is just an example trace and its answer.)
=======
```