Midterm Exam – Graduate Operating Systems CSE 60641 – Fall 2018

NAME: ____

Reminders

1. Make sure to look through entire exam from front to back.
2. You are welcome to use the front and back of the exam pages.
3. Please be concise in your answers if possible. A dissertation response is not needed for most questions.
4. Illustrations can be helpful to convey your point.
5. When in doubt, ask a question.
6. Try to at least have a partial answer for each exam item.
7. Some answers may depend on context. Clearly define the context if necessary for your answer.
8. **The exam is closed book, closed note, closed computer, closed electronic device.**
9. Write your answers legibly.
10. If one of your colleagues missed the exam and has to retake it at a later time, discussing the exam with that colleague before he/she has taken the exam will be considered cheating!

Question 1: Many scalable distributed file system designs separate data servers from metadata servers.

a.) **Provide an example (e.g., describe a specific file I/O scenario) where this is beneficial.**

b.) **Further, many of these solutions use multiple data servers, but just a single metadata server. Why?**

Question 2: The Berkeley Fast File System (FFS) made several changes to the original UNIX file system. One of them was to replace the "free list" with a "bit map".

a.) **Why did FFS implement this modification, i.e., describe at least one significant advantage of a bit map over a free list?**

b.) **Assume that the table below shows a snapshot of a bit map for 64 blocks of data, each block being 1Kbyte. Assume a "1" indicates a free block and a "0" a used block. What is the largest contiguous region in memory you could allocate for a new file? If a new file was 12Kbytes, which blocks would you allocate to maximize contiguity (clearly indicate in the figure below which blocks you would allocate)?**

0	1	0	0	0	0	0	0
1	1	0	1	1	1	1	0
0	0	0	0	0	0	1	1
1	1	1	1	1	1	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	1

Question 3: In Sprite LFS (Log-Structured File System), all data and metadata are grouped into chunks (called "segments") and then written to disk to a free location on disk. Does Sprite LFS try to improve primarily READ or WRITE performance? How is this achieved?

Question 4: Monitors also provide condition variables with two operations: wait and signal (e.g., a program can call wait on condition variable x using x.wait). Below are code segments for two threads (Thread 1 and Thread 2). Show how you can use one (or multiple) condition variable(s) to ensure that the output of the program will always be "Notre Dame beat Virgina Tech!"

Thread 1:

```
printf("Notre");


printf(" beat");
```

Thread 2:

```
printf(" Dame");


printf(" Virginia Tech!");
```

Question 5:

a.) **Explain why RAID 0 (striping) increases read performance, but decreases fault tolerance?**

b.) **Explain how RAID 1 (mirroring) increases both read performance and fault tolerance (surviving any single fault) at the cost of doubling disk usage.**

Question 6: Assume a file system that uses blocks of size 4MB. What would be an advantage of such a large block size? How much internal fragmentation will each file have on average? Now further assume that the system allows you to perform write operations with a granularity of 4KB (instead of the entire 4MB block). What advantage would this have?

Question 7: Explain the concept of "whole-file caching" and what are one advantage and one disadvantage of this approach?

Question 8: You are tasked with building a server where it is more important to service some requests with an upper bound on response latency as opposed to serving all requests with unbounded latency. How would you design such a server?

Question 9: Is it possible for the two threads to deadlock by concurrent calls to the functions *A* and *B* below? If so, give an example of an execution trace that leads up to deadlock. If not, explain why deadlock is impossible.

```
void A (){
        ...
        pthread_mutex_lock (&M2);
        pthread_mutex_lock (&M1);
        ... critical section ...
        pthread_mutex_unlock (&M1);
        pthread_mutex_unlock (&M2);
}
void B (){
        ...
        pthread_mutex_lock (&M1);
        pthread_mutex_lock (&M2);
        ... critical section ...
        pthread_mutex_unlock (&M2);
        pthread_mutex_unlock (&M1);
}
```