Mauricio Verduzco Chavira
A20552976

December 4th, 2023

Part A: Deadlocks

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
// These two functions will run concurrently.
void* print_i(void *ptr) {
 pthread_mutex_lock(&mutex1);
 pthread_mutex_lock(&mutex2);
 printf("I am in i");
 pthread_mutex_unlock(&mutex2);
 pthread_mutex_unlock(&mutex1);
}
void* print_j(void *ptr) {
 pthread_mutex_lock(&mutex2);
 pthread_mutex_lock(&mutex1);
 printf("I am in j");
 pthread_mutex_unlock(&mutex1);
 pthread_mutex_unlock(&mutex2);
}
int main() {
 pthread_t t1, t2;
 int iret1 = pthread_create(&t1, NULL, print_i, NULL);
 int iret2 = pthread_create(&t2, NULL, print_j, NULL);
 while(1){}
 exit(0);
}
```

This code creates two threads. Then the function print_i is trying to lock mutex1 and then mutex2. At the same time print_j is trying to lock mutex2 and then mutex1. When these functions run on different threads the first one locks the first mutex and starts waiting for the mutex2 to be released. The second function in locking the second mutex and waiting for the first one to be released. This means that both functions are withholding the resources the other one needs, not allowing either one of them to finish. This never-ending loop is called deadlock and is what causes the program to not run properly. To fix this we could try to change the order or the locks to be consistent and to avoid the deadlock. Also creating some timeouts for the functions to release or something of the sort.

Mauricio Verduzco Chavira
A20552976

December 4th, 2023

Part B: Discs

Chapter 37 Homework

1. Compute the seek, rotation, and transfer times for the following sets of requests: -a 0, -a 6, -a 30, -a 7,30,8, and finally -a 10,11,12,13.

2. Do the same requests above, but change the seek rate to different values: -S 2, -S 4, -S 8, -S 10, -S 40, -S 0.1. How do the times change?

    a. It goes shorter with a default of 1.

3. Do the same requests above, but change the rotation rate: -R 0.1, -R 0.5, -R 0.01. How do the times change?
    a. Rotate and transfer take longer.

4. FIFO is not always best, e.g., with the request stream -a 7,30,8, what order should the requests be processed in? Run the shortest seek-time first (SSTF) scheduler (-p SSTF) on this workload; how long should it take (seek, rotation, transfer) for each request to be served?
    a. FIFO order: 7, 30, 8
    b. SSTF order: 7, 8, 30

5. Now use the shortest access-time first (SATF) scheduler (-p SATF). Does it make any difference for -a 7,30,8 workload? Find a set of requests where SATF outperforms SSTF; more generally, when is SATF better than SSTF?
    a. It does not make any difference.
    b. The SATF outperforms SSTF when seek time is shorter than rotate time.

6. Here is a request stream to try: -a 10,11,12,13. What goes poorly when it runs? Try adding track skew to address this problem (-o skew). Given the default seek rate, what should the skew be to maximize performance? What about for different seek rates (e.g., -S 2, -S 4)? In general, could you write a formula to figure out the skew?
    a. $Skew = track\_distance/seek\_speed * rotation\_speed/rotational\_space\_degrees$

7. Specify a disk with different density per zone, e.g., -z 10,20,30, which specifies the angular difference between blocks on the outer, middle, and inner tracks. Run some random requests (e.g., -a -1 -A 5,-1,0, which specifies that random requests should be used via the -a -1 flag and that five requests ranging from 0 to the max be generated), and compute the seek, rotation, and transfer

times. Use different random seeds. What is the bandwidth (in sectors per unit time) on the outer, middle, and inner tracks?

a. One
    i. outer: 3/(135+270+140)=0.0055
    ii. middle: 2/(370+260)=0.0032
b. Two
    i. outer: 2/(85+10)=0.0211
    ii. middle: 3/(130+360+145)=0.0047

8. A scheduling window determines how many requests the disk can examine at once. Generate random workloads (e.g., -A 1000,-1,0, with different seeds) and see how long the SATF scheduler takes when the scheduling window is changed from 1 up to the number of requests. How big of a window is needed to maximize performance? Hint: use the -c flag and don't turn on graphics (-G) to run these quickly. When the scheduling window is set to 1, does it matter which policy you are using?
    a. To maximize performance, we need a disk size of 1 which would be somehow equivalent to a kind of FIFO policy.

9. Create a series of requests to starve a particular request, assuming an SATF policy. Given that sequence, how does it perform if you use a bounded SATF (BSATF) scheduling approach? In this approach, you specify the scheduling window (e.g., -w 4); the scheduler only moves onto the next window of requests when all requests in the current window have been serviced. Does this solve starvation? How does it perform, as compared to SATF? In general, how should a disk make this trade-off between performance and starvation avoidance?
    a. SATF: 555
    b. BSATF: 525

10. All the scheduling policies we have looked at thus far are greedy; they pick the next best option instead of looking for an optimal schedule. Can you find a set of requests in which greedy is not optimal?
    a. `-a 9,20 -c`
    b. `-a 9,20 -c -p SATF`

Mauricio Verduzco Chavira
A20552976

December 4th, 2023

Part C: RAID and not for Ants

Chapter 38 Homework

1. Use the simulator to perform some basic RAID mapping tests. Run with different levels (0, 1, 4, 5) and see if you can figure out the mappings of a set of requests. For RAID-5, see if you can figure out the difference between left-symmetric and left-asymmetric layouts. Use some different random seeds to generate different problems than above.

2. Do the same as the first problem, but this time vary the chunk size with -C. How does chunk size change the mappings?

```
0  2  4  P
1  3  5  P
8 10  P  6
9 11  P  7
```

3. Do the same as above but use the -r flag to reverse the nature of each problem.

4. Now use the reverse flag but increase the size of each request with the -S flag. Try specifying sizes of 8k, 12k, and 16k, while varying the RAID level. What happens to the underlying I/O pattern when the size of the request increases? Make sure to try this with the sequential workload too (-W sequential); for what request sizes are RAID-4 and RAID-5 much more I/O efficient?
    a. The much more efficient goes with 16k.

5. Use the timing mode of the simulator (-t) to estimate the performance of 100 random reads to the RAID, while varying the RAID levels, using 4 disks.

6. Do the same as above but increase the number of disks. How does the performance of each RAID level scale as the number of disks increases?
    a. It increases the performance goes about 170%.

7. Do the same as above but use all writes (-w 100) instead of reads. How does the performance of each RAID level scale now? Can you do a rough estimate of the time it will take to complete the workload of 100 random writes?
    a. It went to about 100%. To estimate it

8. Run the timing mode one last time, but this time with a sequential workload (-W sequential). How does the performance vary with RAID level, and when doing reads versus writes? How about

varying the size of each request? What size should you write to a RAID when using RAID-4 or RAID-5?

a. This one go to about 35%.
b. The best one here is 12k.

Mauricio Verduzco Chavira
A20552976

December 4th, 2023

Part D: Fsck

Chapter 42 Homework

1. First, run fsck.py -D; this flag turns off any corruption, and thus you can use it to generate a random file system, and see if you can determine which files and directories are in there. So, go ahead and do that! Use the -p flag to see if you were right. Try this for a few different randomly-generated file systems by setting the seed (-s) to different values, like 1, 2, and 3.

2. Now, let's introduce a corruption. Run fsck.py -S 1 to start. Can you see what inconsistency is introduced? How would you fix it in a real file system repair tool? Use -c to check if you were right.
    a. There is an inconsistency in the 13th bit.

3. Change the seed to -S 3 or -S 19; which inconsistency do you see? Use -c to check your answer. What is different in these two cases?
    a. Seed 3: the inodes says we have two references, but the data only shows one.
    b. Seed 19: the opposite as seed 3, we have two references but inode says its only one.

4. Change the seed to -S 5; which inconsistency do you see? How hard would it be to fix this problem in an automatic way? Use -c to check your answer. Then, introduce a similar inconsistency with -S 38; is this harder/possible to detect? Finally, use -S 642; is this inconsistency detectable? If so, how would you fix the file system?
    a. Seed 5: linked y.
    b. Seed 38. Wp changed its name to wp. Is very hard to detect.
    c. Seed 642: g went to w. We can add numbers as 1 to the names of the folders maybe.

5. Change the seed to -S 6 or -S 13; which inconsistency do you see? Use -c to check your answer. What is the difference across these two cases? What should the repair tool do when encountering such a situation?
    a. Seed 16: the 12th inode doesn't have references. This is a directory.
    b. Seed 13: the 10th inode doesn't have references. This is a folder.
        i. To fix this maybe we could delete something that's going "extra".

6. Change the seed to -S 9; which inconsistency do you see? Use -c to check your answer. Which piece of information should a checkand-repair tool trust in this case?
    a. The m file was changed to a directory.
    b. The information that could help us is the data block itself.

Mauricio Verduzco Chavira
A20552976
December 4th, 2023

7. Change the seed to -S 15; which inconsistency do you see? Use -c to check your answer. What can a repair tool do in this case? If no repair is possible, how much data is lost?
    a. The "/" was changed to a file.
    b. To fix this we could try changing the inode to directory type.
    c. If we do not manage to solve this, all would be lost.

8. Change the seed to -S 10; which inconsistency do you see? Use -c to check your answer. Is there redundancy in the file system structure here that can help a repair?
    a. Seed 10: w was moved.
    b. Lets check the data blocks to look where was w moved. Then the repair will be done in the inode.

9. Change the seed to -S 16 and -S 20; which inconsistency do you see? Use -c to check your answer. How should the repair tool fix the problem?
    a. Seed 16: The data block of the m file was moved to a different empty block.
        i. To fix this we could delete the file.
    b. Seed 20: The data block of the g file was changed to an empty block.
        i. We will go look around the data blocks to and see which is directory and pointing to g.