COMP 3000 A2
Prepared By: Imran Gabrani-Juma
Prepared For: Professor Anil Somayaji
Course: COMP 3000
SN: 101036672

1. Assume you have a file A. You type `ln A B` in order to create file B. What is the relationship between the return of lstat() on A versus B? Explain briefly.

In the file represented we can see that there is a relationship between both file type A and file type B, this is because when we analyze the file, we can see that they both share the same inode. When looking deeper, we can see that this is a result of file type B being a hark link of file type A. Normally, this means that file type B is a copy name that points to the specific Inode. However, one thing that I noticed was that both file type A and file type B are independent of each. What this means is that if we make a change in file type A but didn't want to change file type B, they won't affect each other. They do not make the communication as they are hardlinked. This is why it is important that they are hardlinks and why this tool is important

2. Assume you have a file A. You type `ln -s A B` in order to create file B. What is the relationship between the return of lstat() on A versus B? Explain briefly.

When looking at this question, it is important to be aware that both file type A and file type B both have different inode numbers. This preliminary indication is important to know as this question is very similar to the relationship seen above. When looking at the relationship we can see that file type B is currently pointing to file type A. This representation is what is often described as a symbolic link. We know from definition that a symbolic links is when the file copies the path of another file type. This case is represented when file type B mirrors the path of file type A, however, similarly to our answer from the previous question, since both the file types that we are working with are linked, a change in one **will** show In the other unlike how it was hardlinked in question 1. This concept is a result of the action when we read the file type or write within the file type, it is using the identical file descriptor to compile the task. Thus, changes will be the same and mirrored across the file types.

3. How could you modify 3000test.c so it can report on what a file's user ID and group ID are and the corresponding username and group name? Specify the changes you make rather than writing all the code out. (In other words, code a solution and explain the changes you made here.)

When looking at the 3000test file, we can see that this process already calls the stat () function, this call is used so that we can gain information about the file in question. When stat() is called, it gives the information we need about the inode and from here we can get the correct structure. However, if we need the User ID and the Group ID, it is most efficient to use statbuf.st_uid and statbuf.st_gid. This will access the correct information that we need. During the access we will also need to use the function call of getpwuid, from here we will be able to return the correct information we need. .

4. If you change line 68 of 3000test.c to be `data[i] = 'A';` (from `count++;`), what will the program do? Explain briefly.

When looking at this line in the 3000test.c file it becomes evident that if we change the script from ‘a’ to A, we will make the script less efficient and it will not complete the task to what we originally designed it to do. If we read through the whole script we can see that the purpose of this line is to count how many characters of that are in the specific file type. However, if we change this notation we will see that it is now counting another **KIND** of script and not what we want it to do. This will now look for capital in the file and not lower case defeating what the task it was indented to do.

5. What does setup_comm_fn() do in 3000shell?

Looking at the 3000shell file, we can see that setup_common_fn() will create a place for the file type in the correct path. However, if we do not have this function, we will not see the correct information and what is displayed will be very minimal. This will allow us to use the path in the correct order for the plist to get the process and see what is currently being ran.

6. How can you modify 3000pc so the consumer permanently stops consuming once it finds an empty queue?

When looking at the 3000pc file, we can see that we can manipulate the file many different ways to complete the task that is being requested. Looking at the code, the most efficient way to permanently stop would be to look within the get_next_word function. When sitting with the TA from the course he directed me to look within the loop however I could not find the correct solution to this problem.

7. What would happen to 3000pc if we replaced lines 341-347 with `s = (shared *) malloc(sizeof(shared));`? Why?

When we look at the 3000pc file, and the after affects we can test this and see that there would be no significant affect to the program. When experimenting with the process test results showed that the program was running at the normal speed, as well as input size was also following the same protocol and making itself less or equal to the que size. Majority of the time, we noticed that if the process was greater than the que size the program would enter it self into a limp mode and was running inefficiently. After further discovery it seemed evident within the mmap call, the flag of MAP_SHARED and MAP_ANONYMOUS that was allowing the process to continue. When understanding the 3000pc file it is important to note that without the signal directing the program what to do, the limp mode or hang will continue until the program is killed because the process is waiting for a signal that it cannot produce at this time.

8. In 3000pc, what happens if you remove the calls to kill()? What does this tell you about the roles of signals in this program?

When looking at the 3000pc file, we can see that this process is heavily dependent on many other processes thus if we are to remove the kill() command we will not run the process as efficiently as it is supposed to be running. For starters, when looking at the 3000pc file we can see that this will affect 2 major groups. The producer and the consumer when we remove this command we will be preventing both parties from waking up when the program goes to sleep, as a result when the consumer decides to go to sleep it will have to wait for the time interval to pass before it can wake up the producer. Likewise, when the producer goes to sleep it will have to wait till the time passes to wake up the consumer. Here, the role of the signal is to make the program run much more efficient so you don't have to wait for the time interval to pass between when the processes go to sleep.

9. Is it possible to predict the output of /dev/random? Explain briefly.

When looking at this question, I believe that It is false, we cannot predict the outcome of the dev/random process. This is because when we are using a linux system, the random key will generate a series of random numbers that are created on the entropy pool. What this means is the pool will generate a series of random values by taking in the input from the various keys entered or movements by accessory devices and then take those values in. From this, it is virtually impossible to predict the output as you can't even look at the preliminary data that is being presented. Another indication that we cannot predict this is when you look at the process you can see that it does take some time to generate the values, this is because the size of the pool is almost infinite and once it has reached its maximum capacity the system will fill the pool again. Thus, we cannot predict the output.

10. Do sem_wait() and sem_post() generate system calls? (Be sure to check!) Why?

When looking at the function types, I could not find a complete answer for this question, however when I went to office hours the I was shown that they do not generate any system calls between the two.