COMP 3000 A4
Prepared By: Imran Gabrani-Juma
Prepared For: Professor Anil Somayaji
Course: COMP 3000
SN: 101036672

1. **[1] Does `alloc_pages()` in the Linux kernel immediately allocate memory, or is memory allocated when it is accessed?**

   When applying this process “Alloc_pages” will directly allocate the memory immediately

2. **[1] Does an anonymous mmap system call allocate memory immediately, or is the memory allocated as it is accessed?**

   When this system call is put into place, the anonymous mmap will allocate the memory however this will only occur as it is accessed

3. **[2] What is one key advantage of lazy memory allocation? What is one key disadvantage?**

   The main advantage of this process is that the process does not use unnecessary resources unless they are absolutely needed, thus meaning it will not take up a specific resource unless it is required. One of the main advantages of this is that we can significantly decrease the boot time of the program that we are trying to execute, as well in some special cases we can directly eliminate the allocation completely, this is a result of it not needing it in the first place. On the other hand, one of the main disadvantages is that if you do require memory. Is will case the process to become a lot less efficient as the we will be running a lot slower, this happened because the program has to wait until the program has memory to allocate it before it can start to use it.

4. **[2] Assume we have modified the remember module so it allocates 16K (4 pages) at a time, and assume we have stored in /dev/remember 16K copies of the letter "a".**

   - **As written, what will a read of /dev/remember return if we use a 4K buffer, starting at the beginning of the file?**

     When completing this task, is it best that we refer back to our tutorials that we completed. We can recall that since the buffer that we are using is only 4K and we are attempting to read 16K a’s we must give it back to the buffer as it will return a small error.

   - **What will happen if we do a second read, again with a 4K buffer (on the same open file)?**

     For this task, if we tired to reattempt the task, we would see the same result as we did from the previous attempt. This is a result due to the fact that the buffer that we currently have is not big enough to hold all of the a’s that we need. The error that we will receive during the second attempt will be the same error as we received in the first attempt.

5. **[2] When is memory dynamically allocated in the remember module? When is it freed?**

When looking at this problem, and from the problems from the previous assignments we know that when memory is being dynamically allocated and deallocated, both tasks will occur in the write function of our process. However it is important to note that memory can also be deallocated in the exit function of the process as well, similar to the we can see that we can allocate memory will execute when we want to save the data that we have written when the system call of init_save_data is process where the alloc_page is made. Once these processes are both completed, the memory can be freed once we run the free_saved_data function in the process.

6. **[2] As written, what happens when multiple write calls are made to /dev/remember? What is remembered? Why?**

When the system calls are made to /dev/remember we will allocate the specific memory that we need and then save it, thus when the next call is made to write, and free the old memory, we will be able to allocate the new memory and then save it the same way. This allows us to remember the last call that was written and then every new call that is made willl erase the previous call that we made almost in a looping method if you are to visualize it

7. **[2] What system call causes getattr() in memoryll.py to be called (on files in a memoryll filesystem)? How did you verify this?**

When observing this problem, we can see that the system call will launch getattr() in the memory.py process to be called in a form of fstat. When this process is launch in the VM I observed that the process created an strace of the entire program.

8. **[2] What function in memoryll would you have to change to make it impossible to change file timestamps? Why?**

When looking at this particular question, it is almost virtually impossible to do this kind of task, however what way that you could do it would be to change the setattr() function within the program. This allows the process to commute as when we access the file inside of the mnt process we can see that we receive a message displaying the following output,

“[setattr: 2[‘st_atime’, ‘st_mtime’]]”

What this message is telling us, is that the process is attempting to change the time stamp of the file in question.

9. **[2] When you run fusermount via execve, what euid does fusermount's process have? What uid does it have?**

In this particular task, when you the executer the process of fusermount, we can see that euid will contain a value of 0, this is a result of the root directory that it is referring to. As well. The we must note that the uid is also 0 in this case.

10. **[2] What is a process's effective uid (euid)? Why is the euid not always equal to its uid?**

When looking at this problem from the lecture component on the class we have been taught that they are not alway equal to the uid as a result of the kernel using the euid to determine what kind of privileges the current process will have access to as well this is also a factor of when analyzing the code, we can see that we have a system call of setuid or setgit, when we have these kinds of system calls pre sent we can see that the processes euid is change to whatever the executable files uid or gid, will be. An important aspect of this question is note that the uid and gid will always remain the same as it was prior to when the execve was called.

11. **[2] What system calls does memoryll make when performing file operations on behalf of the kernel? Give one concrete example showing how things work. (Hint: you won't be able to strace memoryll as a regular user.)**

This question recalls previous knowledge from the tutorials, here we see what we do require a much higher clerance or permission to strace the program. As a result one way that we can get around this problem would be to do a

“sudo strace python memoryll.py newmnt”

After running this piece of script, we can use the command tail -f on the file that we are calling the system calls on. This will allow us to display multiple calls to read, fstat, write, openat, and the writeey members.