COMP 3000 A1
Prepared By: Imran Gabrani-Juma
Prepared For: Professor Lou Nel
Course: COMP 3000
SN: 101036672

1. **What does it mean to run a program "in the background"? Specifically, what is the key difference between running a program in the foreground and in the background?**

   When running a program in the foreground, for example a program like Mac OS we have to wait for the task to finish loading, before we can request it to make multiple tasks. Likewise, when running a program in the background we can have multiple tasks being launched as long as we open a new terminal. One thing to keep in mind is that if end the terminal than all tasks will be gone and exited once we quit.

2. **Are system calls used to receive signals? Explain briefly. (Hint: strace a program and send it a signal. What happens?)**

   Precisely, when looking at systems calls they are directly related to receiving signal calls because of the main objective that they are trying to reach. When looking at the process that was explained in tutorials, we demonstrated that the system call was dependent on the signal to confirm if the process was completed. During tutorial, we referenced that the 3000shell did this exact thing. For example during the tutorial, we were required to use system call of "RT_SIGACTION" to complete this process in the 3000shell given, by doing this we are able to physically prove this theory as the signal was waiting for the action before allowing us to complete another task in the shell.

3. **The system calls 3000shell uses to search for a program to run (in one of the PATH directories) are not the same as those used by standard shells such as bash. What's the difference?**

   The only logical way to see if this was possible was to use the sample program and see what we could find, with this I am able to confidently determine that by running the bash we are able to achieve a lot more, in a more efficient way. As-well, the way that we were able to do this was by running the 3000shell and running the strance command on it. Once that was done, we did the same process on the ls command that displays all the files in the bash. By running the strance on both commands we are able to come to this conclusion that the bash is better.

4. **How could you change 3000shell so that it generates zombie processes?**

   When looking at exactly what a zombie process is, by definition we can see that it is a process that cannot closed or “exited” however it has finished executing its tasks. Normally, we can see that a zombie process will occur when we have both a parent and child process both running at the same time. However, the child process cannot communicate this to the parents that it has completed its process, so we are stuck in what is called a zombie process. Unfortunately, the only way that we can get out of the zombie process loop is to send a SIGKILL command within the terminal. This will subsequently end the part process. As-well if we are looking for a method to create a parent process, this can be done within the provided 3000shell program written in C. We can easily do his by commenting out the signal handler that has been put into place and then we will have to wait on the main function call.

5. **How would the behavior of 3000shell change if line 286 was removed? (pid = fork();) Why?**

   When looking at line 286, we can see that if we remove pid = fork(); we will notice that the parents and child process will no longer be in the program. Thus creating a chain of events, primarily leading to, a process that when the program is executed we will have no problems running the first task however once we have inputted our first argument into the terminal we will exit the program. The reason that this happens is because the command is called to kill the process once the process has been done running

6. **In 3000shell, when are lines 299 and 300 executed? Why?**

   When observing these line, we can see that there is an if statement that has been placed here to make sure that the “If statement” that the user has inputted is a valid line into our terminal. This is almost done as a QA (Quality Assurance) line to make sure that everything is kept running smoothly. As-well, this line does not act as a replacement if the user decides to change the directory by entering the command “cd”. One thing that we need to keep in mind is that if you decide to change the directory, and then use an if command, the statement will still be executed the same way by printing the error message.

7. **Does using getenv() generate any additional library calls (as reported by ltrace)? Does it generate any additional system calls (as reported by strace)? Why?**

   When looking at this questions we can see that by fact, ltrance will indefinitely make less calls, this was seen by looking at the reports that were created in the tutorial, however when we choose to ignore the sigaction and the printf library cal, there are only two other possible calls that can me made. As-well, we can see that strace does not report any additional system calls when looking at this. The only other calls that we can see here are getenv. One reason that this happens is because the find_env function looks though the whole system and will compare the value that is seen to the value of the function or string that was passed though. From this we can come to the conclusion that every loop there is another call that will come as a result of the library call.

8. **Does parse_args() allocate any memory on the heap (i.e., any memory that stays allocated after the function returns)? How do you know? Give a brief argument.**

   When observing the code that is presented here and the functions that are included we can see that we have no function or system calls that are related to calloc or malloc. This would mean that there is no memory or memory that is being allocated within the heap in this function program. Consequently, one method that I was taught in a previous course that we could use to double check this theory would be to use mtrace within the system. When I did this, I was given the same conclusion that there were no memory leaks within the system and program. With this we can confidently suggest that there is no memory allocated, nor that there is no memory leaks within the heap presented here.

9. **execve overwrites all of a process's memory with that of a new executable. Does execve also close all open file descriptors? How do you know (from your experience with 3000shell)?**

   In the time that we have been working with the 3000shell we can confirm that the file descriptors will always remain open, this is because they are passed onto the new process when the call is execve. This happens because for example, if they were closed it would basically corrupt the other files that are also being hosted. When looking deeper, we can identify that there are three files that are kept open to point to the other files and their status within the program. However, once these are removed we would have major disruption as we cannot identify these anymore. However, because this did not happen it means that these files where kept open and just translated. Meaning they where passed onto the next process in the 3000shell.

10. **What happens to an in-progress system call when a process receives a signal? (An example is a program waiting for input from a terminal with a blocking read call.) What does "restarting" a system call have to do with this?**

    When looking at this question, we can see that the system call that we are looking at gets passed and interrupted by the signal. This will stop what the program was originally trying to execute and will subsequently cancel the system call that is trying to be made. This has a positive affect so it will have no system affect on the program making it seem as if there was no call every made to disrupt the program.

11. **If you changed plist() so it took proc_prefix as an argument (rather than accessing it as a global variable), what output would it produce when given an argument other than "/proc"? Explain briefly.**

    When I tried to run this process I was presented with the follow message "Error, could not open /proc. When doing some more investigation with the code, we can see that the name /proc is actually the individual file within the system. It's main function is to store the information about the system and to store the information about the process within the stem. However, if you also try it to change the file name that Is give here you will be presented with another error as the file won't have any of the correct information about the system itself or the process that it is trying to run.

12. **Describe how you could add output redirection for external programs to 3000shell.**

    When looking at this problem, there are many different aspect that must be completed before we can complete this task properly. The first task that we need to find the character pointer that has the requirements of what we would like to import into the file. Once we have this, we must make a unique file descriptor with an identifier. By creating this we will be able to reference the file that we want to create with the correct outputs. Once we can created this, our next step is to write a script that will allow us to open the file that we have created, this will also need to have the correct permission within the file otherwise this will not work. To get these, I suggest that we use the system call of 'SYSCALL' to write the file we want. Now that we have these two parts, we must create a few different parameters, firstly, we will need one that will have the system call that we will looking for, then we will need to specify the exact file that we are looking to write, finally we will need the character pointer to point to the message, with this it will be able to take in the correct value of bytes for the message to be taken in. Once this process is complete, we will need to have the script close the file so that we don't get any outside interference within the file. Should this happen it will have a negative result. Once this is completed we should be able to see the output in the file that we have created. If all is done correctly it should be placed in a readable file so that way we have no issues when we want to go back and later see the data we have resulted.