Lab Exercises - I

1. Write a program that creates two child processes. One child process should print its process ID (PID), and the other child process should print its parent process ID (PPID). The parent process should wait for both child processes to terminate before ending. **Hint: fork (), wait ()**

2. Create a program that creates a child process. The child process prints "I am a child process" 100 times in a loop. Whereas the parent process prints "I am a parent process" 100 times in a loop.

3. Develop a program that prints the current process ID (PID) and the parent process ID (PPID) of the running process. Additionally, implement a function to retrieve and print the user ID (UID) of the current user executing the program. **Hint: getpid (), getppid ()**

4. Create a program named stat that takes an integer array as command line argument (deliminated by some character such as $). The program then creates 3 child processes each of which does exactly one task from the following: i) Adds them and print the result on the screen. (done by child 1), ii) Shows the average on the screen. (done by child 2), iii) Prints the maximum number on the screen. (done by child 3)

5. Create a program that reads input from a text file named "input.txt" and writes the content to another text file named "output.txt". Ensure proper error handling for file opening, reading, and writing operations. **Hint: open (), read (), write (), close ()**

6. Invoke at least 4 commands from your programs, such as Cp, mkdir, rmdir, etc (The calling program must not be destroyed)

7. Write a program that demonstrates the usage of the fork() system call to create a child process. The child process should execute a shell command to list all files in the current directory, while the parent process waits for the child to terminate. Ensure proper synchronization between parent and child processes. **Hint: fork (), execlp ()**

8. Implement a program that utilizes the sleep()` and `alarm()` system calls. The program should initially set an alarm for 5 seconds and then enter a sleep loop. Within this loop, the program should print a message every second. When the alarm signal is received after 5 seconds, the program should terminate gracefully. **Hint: sleep (), alarm ()**

Lab Exercises - II

1. Write a program which uses fork () system-call to create a child process. The child process prints the contents of the current directory, and the parent process waits for the childprocess to terminate.

2. Write a program which prints its PID and uses fork () system call to create a child process. After fork () system call, both parent and child processes print what kind of processthey are and their PID. Also, the parent process prints its child's PID, and the child process prints its parent's PID.

3. Develop a program that copies the contents of one file into another file using system calls. The program should accept two file paths as command-line arguments: the source file to be copied from and the destination file to be copied to. Ensure proper error handling for file opening, reading, and writing operations.

4. Write a program that lists all files and directories in the current directory using system calls. The program should traverse the directory structure recursively and print the names of all files and directories found, along with their respective types (file or directory).

Solution hints for problem # 4:

1. Useful System Calls:
   - opendir(): Use this system call to open a directory stream for the current directory.
   - readdir(): Call this system call to read the next entry from the directory stream opened by opendir().
   - closedir(): Use this system call to close the directory stream after reading all entries.

2. Looping through Directory Entries:
   - Use a loop to iterate through each directory entry returned by readdir().
   - Check the type of each entry using the st_mode field of the struct dirent structure returned by readdir().

3. Handling Subdirectories Recursively:
   - For each directory entry that is a directory itself, consider recursively calling the same function to list its contents.
   - Ensure proper termination conditions to avoid infinite recursion.

4. Printing File and Directory Names:
   - Print the names of files and directories using the d_name field of the struct dirent structure returned by readdir().
   - Determine whether an entry is a file or a directory by checking its type using st_mode.