LAB #6

Name	ID
Fairs khader al-Ghamdi	1847152

2) Are the process ID numbers of parent and child threads the same or different? Why?

The parent and child threads within the same process share identical process ID numbers. This occurs because in a multithreaded process, both the parent and child threads utilize the process resources, including memory and file descriptors, leading to the same process ID. Nonetheless, each thread possesses a distinct thread identifier, signifying their individuality as separate threads executing within the shared process.

3) Does the program give the same output every time? Why?

The program does not produce consistent output each time it runs. This inconsistency arises from the fundamental characteristics of multithreading, where the order of thread execution lacks determinism. The threads operate concurrently, and the specific sequence in which they execute is determined by the thread scheduler of the operating system, which can vary between program executions. The presence of a shared global variable, glob data, in both the parent and child threads results in a race condition. This means that the final output is dependent on factors such as the timing of thread execution and the occurrence of context switches between threads.

4) Does the threads have separate copies of program data?

No, glob_data is not duplicated for each thread. In a multithreaded program, all threads utilize the same global memory, which encompasses shared entities like the global variable glob data.

5) Do the output lines come in the same order every time? Why?

No, the order of output lines is not consistently identical each time. This occurrence arises due to the non-deterministic nature of thread execution, which is determined by the operating system's thread scheduler. The thread scheduler considers various factors such as system load, thread priority, and

other variables that can influence the order in which threads are scheduled to run. When multiple threads are created, as demonstrated in this program through a loop, there is no assurance or guarantee regarding the specific sequence in which they will be scheduled for execution.

6) Are the local addresses the same in each thread? What about the global addresses?

The local addresses differ across threads as each thread possesses its own stack space. Consequently, the address of local_thread varies for each individual thread. However, the global addresses remain consistent across all threads. This is attributable to the fact that all threads share the same global memory, resulting in this_is_global having an identical address across all threads.

7) Are the local addresses the same in each process? What about global addresses? What happened?

No, the values of local_main and this_is_global variables in the parent process remain unchanged after the completion of the child process. This is due to the behavior of the fork system call, which creates a new process with a separate copy of the entire memory of the parent process. Therefore, when the child process modifies its own copy of local_main and this_is_global, these changes do not impact the values of those variables in the parent process's memory.

8) Are the local addresses the same in each process? What about global addresses? What happened?

At the moment of the fork operation, both the local and global addresses are identical in each process. This behavior is a result of the underlying mechanism of the fork operation in Unix-based systems. When fork is invoked, the operating system generates a nearly identical replica of the current process, including a duplicate of its memory layout. However, it is crucial to understand that these are distinct copies of memory, and modifications made in one process do not impact the memory of the other process. Therefore, despite the visual similarity of memory addresses, they

represent separate and isolated memory segments in the child and parent processes.