Title: Operating Systems Assignment 1 2021-2022
Full Name: Damianakis Damianos

In the code, there are several explanatory comments regarding the implementation. The assignment is implemented in C++ using the g++ compiler.

Inside the zip file, there are two text files - a small and a large one - for testing the code. The values I've provided for demonstration are based on the "small_sample.txt" input file, with 5 children and 10 dosages. You are free to test larger values and the larger file ("big_sample.txt").

The project consists of two CPP files: "main.cpp," where the main function resides, and "func.cpp," containing necessary functions for the assignment's implementation. Additionally, there's a header file with appropriate library includes and function prototypes. A "makefile" is also included (explained below).

Commands:
- `make clean`: Deletes all executable ".o" files.
- `make`: Compiles the program with necessary flags.
- `make run`: Executes the program. Arguments (file name, number of children, number of dosages) are provided in the makefile, allowing you to set the file and numbers for testing without needing to recompile.
- `make valgrind`: Runs Valgrind to check for memory leaks during program execution.

Implementation Approach:
1. Initially, the code checks if the user has provided the correct arguments (from the makefile). A function has been created that takes a file as an argument and returns the number of lines, enabling calculation of the input file's lines.
2. Two semaphores, defined as global, are initialized above the main function. Using `sem_post`, the value of semaphore 2 is decreased by one.
3. Shared memory is created using a function that returns the mmap.
4. The input file is read using the `FileReading` function, which returns an array of the file's lines.
5. Within a loop running for the number of children, child processes are created using the fork system call.
6. Child code: Within an internal loop running for the number of dosages, semaphore 2 is locked using `sem_wait()`, and memory is copied using `writeInt`. Subsequently, semaphore 1 is unlocked, and semaphore 2 is locked again. The average time per child for each dosage is calculated using the clock and printed.
7. Parent code: Inside another internal loop running for the number of dosages, semaphore 1 is locked, and the line requested by the parent is printed. After the loop, the parent waits for "a child to die" using `wait`.
8. Finally, memory is released using `sem_unlink` and delete.