Lab Assignment 2. Processes, Threads and Synchronization Module

It is supposed that some of the SF-format files managed by your previously developed "coordinator.c" program were infected by a Romanian crypto-malware named ANISSM (Aici Nu Invatam Sa Scriem Malware). The malware changed the infected files in the following way:

1. Encrypted some of the binary sections of the file, by performing a XOR operation with an encryption key on each element of the section. The encryption key's length is equal to the length of the encrypted section's elements. Each section was encrypted with a different key.
2. Stored the encryption keys in a new section of the SF-format file, named "SECT_ANISSM", having the type RESERVED. The section's contents actually conforms the TEXT sections' format, with text lines of the form: "ENCRYPTED_SECTION_NAME ENCRYPTED_KEY". The encryption key is stored as a text representation of the binary key, like for example "3FA7" string for the 0x3FA7 hexadecimal value.

You are required to:

1. Add to your "coodinator.c" program support for the following new commands:
   `SCAN_FILE file_name`
   `SCAN_DIR dir_path`
   `CLEAN_FILE file_name`
   The functionality of each command is the following:

   1. (20 min) SCAN_FILE: check if the given file is a SF-format file and is infected with the ANISSM malware. The infection checking must be done based on the specific section name. If the file is infected the message "`file_name: INFECTED WITH ANISSM`" must be displayed on the screen. If the file is not infected, the message "`file_name: NOT INFECTED WITH ANISSM`" must be displayed on the screen.
   2. (30 min) SCAN_DIR: searches the given directory recursively (i.e. the entire file tree starting from that directory) for infected files, displaying on the screen for each checked file a message similar to those displayed by command SCAN_FILE.
   3. (60 min) CLEAN_FILE: checks if the given file is infected and if it is, cleans it by reading the ANISSM section's contents line by line, getting the corresponding section name and encryption key, converts the encryption key from its text representation to its binary representation, decrypts the encrypted section (by XOR-ing again with the encryption key), removes the used line from the ANISSM section and correspondingly updates the section size in its header, removing also the section's header when all its lines are removed (i.e. all encrypted sections were decrypted).
2. In order to improve the performance of the newly added command, you must implement them in the following way:
   1. (30 min) For the SCAN_DIR command, for each file to be scanned a new process must be created. Before displaying its scanning result (message) on the screen, each such a process must get exclusive access to the screen using a lock-like global semaphore, accessible to all implied processes.
   2. (60 min) For the CLEAN_FILE command, the main thread of the process being allocated the file must create a new thread for each section that must be decrypted. It is the responsability of such a section thread to remove its associated line from the ANISSM section and update correspondingly the ANISSM section's size. In order to maintain consistency of the ANISSM section, exclusive access must be get by using a process local lock. The main thread must wait for all section threads to terminate decrypting their allocated section and update the section's size, before removing the ANISSM section's header completely. The synchronization between

the main thread of a cleaning process and the created section threads must be done using locks and condition variables.

IMPORTANT NOTES

1. **YOU MUST UPLOAD ONLY YOUR (updated) "coordinator.c" file!**
2. It is supposed to use your solution to the previous lab assignment. If you did not provide a solution for the previous assignment or the one you provided did not covered all required functionality, you can simply consider all files as SF-format files, hardcode in your program for each file the values for its header fields and focus on creation and synchronization of processes and threads.
3. **Your program must successfully compile with a command line like "`gcc -Wall -Werror -lpthread coordinator.c -o coordinator`" to be further evaluated.** The "`-Wall`" option says the compiler to report anything that could be wrong from its point of view, while the "`-Werror`" says it to report any warning as an error and do not accept it. As a consequence, your program must compile without any error and warning at all in order to be admitted for evaluation.
4. You are required and **restricted to use only the OS system calls**, i.e. low-level functions, not higher-level one, in your entire solution, in all lab assignments. For instance, regarding the file accesses, you **MUST use system calls** like `open()`, `read()`, `write()` etc., but NOT higher-level functions like `fopen()`, `fgets()`, `fscanf()`, `fprintf()` etc. The only accepted exceptions from this requirement are the functions to read from STDIN or display to STDOUT / STDERR, like `scanf()`, `printf()`, `perror()` and functions for string manipulation and conversion like `sscanf()`, `snprintf()`.
5. For testing purposes you can download the following archive, containing a directory with ANISSM infected SF files. Please note that there could be both valid and invalid SF format files in the given archive, though only some of the valid SF-format files are infected.
6. The SF-format files do always start their first section at an large enough offset, such that the ANISSM malware must not change existing sections' offsets.
7. For string tokenization (i.e. separate a string into elements based on specific separators, like spaces) we recommend you using the `strtok()` or `strtok_r()` functions.
8. You can automate your program runs by redirecting its STDIN to a text file prepared to contain the commands normally given from the keyboard. This way you can easily and efficiently run the same test more times for debugging purposes. This is actually the way we will test your solutions.
9. Extra credit (3 points) will be given to you if you write the code of the ANISSM crypto-malware.