Hacettepe University Computer Engineering Department		
Course	:	BBM342 Operating Systems
Experiment No	:	2
Subject	:	POSIX Thread Programming: Simple Cypher
Advisors	:	Dr. Ahmet Burak CAN RA Kazım SARIKAYA
Due Date	:	12 May 2013

INTRODUCTION

Operating systems provide a conceptual model of processes, which run machine codes sequentially. Processes can be created and terminated dynamically, and run concurrently with other processes. Each process has its own address space, which can not be reached by other processes.

For some applications, it is useful to have multiple threads of control within a single process. These threads are scheduled independently and each one has its own stack, but all the threads in a process share a common address space. Because threads share a common address space, they can communicate and share data in this address space.

When two or more processes/threads try to reach same memory address/data, data consistency should be protected and thread/process interference have to be prevented. That means only one process or thread must write/change data in any time. However, data can be read by many processes or threads at the same time. For consistency, if there is any writing operation, the read/write operations should be organized with locks, semaphores and monitors. These primitives are used to ensure that no two processes/threads are ever in their critical regions at the same time.

UNIX allows programming with threads using POSIX thread (pthread) library. Also POSIX library provides several syncronization mechanisms to manage critical sections. To add multithreading capability to your programs on UNIX machines, you should add *pthread.h* to C source code and link the executable with *libpthread* library.

The aims of this experiment are to use pthread library, manage critical sections, and observe performance of thread programming.

Experiment Description

You should implement a simple encryption/decryption program w/o threading. The program should accept three command line parameters.

The first parameter denotes the operation, which can be either encryption/decryption. If it is *e*, program will encrypt the file, if *d*, program will decrypt the file. The second parameter is the name of a file, which should be a binary file. The fourth parameter is the key file.

Firstly you should implement the program without threads. The program should read the input file. In each read operation, the program should read a 256-bits block. Then the program encrypts/decrypts the block and writes the block back to its original position. Be careful, you do not write into a new file. A visualization of an input file during encryption is:

EEEEPPPPPPP...

E means encrypted, P means plain text. Here, the first four blocks are encrypted and remaining blocks are not encrypted yet.

After implementing a single threaded file, you should implement a multithreaded program. You should write one reader thread, four encryptor/decryptor threads, and one writer thread. Reader and writer threads works on 256-bit blocks. However reader thread should write the blocks into a 1024-bytes buffer (input buffer), writer thread should read the blocks from another 1024-bytes buffer (output buffer). The four encryptor/decryptor threads should read blocks from the input buffer, encrypt/decrypt blocks, and write them to the output buffer. The program is illustrated below:

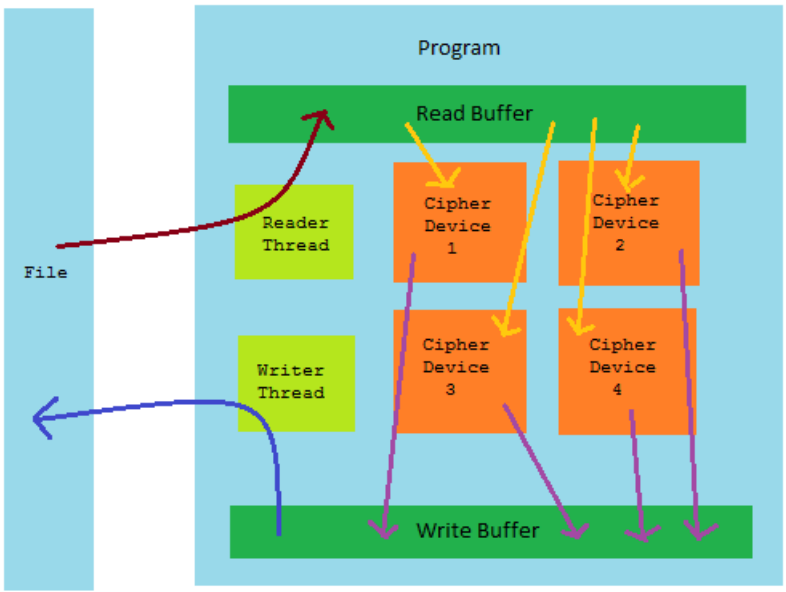

Figure 1: Program

Simple Cypher

For encryption/decryption operations, you should implement a simple cypher algorithm that is a Substitution Permutation Network (SPN). The algorithm should encrypt/decrypt 256-bits blocks. Algorithm is symmetric and uses 256-bits key. The encryption operation on a 256-bit block performs the followings steps 256 times:

1. XOR 256-bits block with the key.
2. Rotate the key left.
3. Substitute 8-bits blocks of the 256-bits blocks.
4. Permute 16-bits blocks of the 256-bits blocks.

In encryption operation, the substitution is to multiply each 8-bits by 137 on modulo 256. The permutation is (6 3 16 11 7 14 8 5 15 1 2 4 13 9 10 12)

The decryption operation on a 256-bit block perform the followings steps (reverse of encryption steps) 256 times:

1. Permute 16-bits blocks of the 256-bits blocks.
2. Substitute 8-bits blocks of the 256-bits blocks.
3. Rotate the key right.
4. XOR 256-bits block with the key.

In the decryption operation, the substitution is to multiply with 185 on modulo 256. In decryption operation, the reverse of encryption permutation is applied.

REPORT

You should explain critical section entrance and exit. You should give the test results for performance of single and multi threaded approaches.

SUBMIT

You should submit your codes from https://submit.cs.hacettepe.edu.tr. You should send your Makefile,single.c,multi.c inside source directory and report.pdf inside report directory. Your code will be evaluated according to execution, implementation, indentation, and comments.

NOTES

- Do not make any performance optimizations over encryption algorithm (Aim of this algorithm is to slow encryption or decryption operations).
- Collaboration is prohibited.
- Downloaded or modified source codes from Internet resources will be considered as cheating.