2023, Spring CS-21 Computer Architecture/Assembly Language Chapter 11 Project: Encrypted Files

Bill Komanetsky
Las Positas College

April 27, 2023

---

1 Project Overview

This will be a combination project of a couple of Chapters. Chapter 8 (stack frames and local function variable) and Chapter 11 (Files and Dynamic memory allocation)

We don't like the Unix-flavored cp command because it really doesn't tell us enough or do enough for us, so, we want to write a file copy program which can take any sized file and copy it as an encrypted file and then decry pt it later using the exact same file/key and command.

- When the user starts the program with the source file, destination file as arguments, you must then prompt the user to enter an encryption key. This key will be the encryption key used by your looped xor encryption algorithm as you did in a previous assignment.
- If the source file is a previously encrypted file from a previous execution of this program, it will do the exact same encryption as previously defined, which will actually decrypt the data in the file to the output file.

This will be a Linux, 64-bit only assignment

Total Points: 60

1.1 Requirement 1

Create an application which takes two command line arguments: Source file and Destination file

- If the user types no arguments, or more than two arguments, display an error message that tells them clearly what the problem is
- Save the addresses of the first and second arguments so you can use them for the file names in the copy

---

1.2 Requirement 2

Create keyboad/display input/output functions

- Create a function, using a stack frame called **inputKeyboard**. This function should take the buffer and buffer length as arguments that will contain the contents of what the user input on the keyboard and the length of this buffer. Use the console keyboard input interrupt as described in the classroom presentation. Within this function you should NOT use my functions.o or functions64.o files or functions. You should use this function to get keyboard input from the user
- Create a function, using a stack frame called **outputDisplay**. This function should take the string array address and the length of the array h as arguments that will contain which will be display to stdout. Use the console output interrupt as described in the classroom presentation. Within this function you should NOT use my functions.o or functions64.o files or functions. You should use this function to output anything to the display.

1.3 Requirement 3

- Open the input file using the first argument. If an error occurs, display an error message using your **outputDisplay** function and end the program
- Open the output file using the second argument. If an error occurs, display an error message using your **outputDisplay** function and end the program
- Display to the user that the source file is being copied to the destination file using your **outputDisplay** function

---

1.4 Requirement 4

- Now, prompt the user for an encryption key. Use your **inputKeyboard** function to do that
- Make sure the key isn't blank. If it is, display an error message and end the program

---

1.5 Requirement 5

Once the number of arguments has been verified and both files opened successfully, dynamically allocate 0ffffh bytes of dynamic memory.

- This will be the work area where data will be read in from the file before encrypting and then writing the encrypted data to the output file

---

1.6 Requirement 6

Now, in a loop, read 0ffffh bytes of data from the file placing it into the dynamically allocated memory obtained in step #5. This will be part of a loop which keeps going until the returned number of bytes read is not equal to 0ffffh bytes.

1.7 Requirement7

Next, encrypt the file-read data found in the dynamically allocated memory with the encryption key entered in #4 by calling a function called ***EncryptMe***. The first argument to this function should be the address of the allocated memory, the second should be the length of the allocated memory, the third should be the address to the encryption/decryption key, the fourth should be the length of the encryption key. Here is how it should work:

1. The function should use a fully-functional stack frame
2. The arguments should be sent to the function as a pushed arguments onto the stack
3. There should be no reason to create any locally created variables
4. The data you encrypt can be overwritten by the encrypted data. In other words, for each byte in the array sent to this function, the same byte can be overwritten by its encrypted counter-part. You do not need to allocate a temporary memory area to hold the encrypted data - you can use the same memory area where data from the file was read into
5. This encryption should work just like the string encryption done in a previous project

---

1.8 Requirement 8

After encryption, write out the number of bytes read from the input file to the output file

- The exit condition to get out of the loop will be if the total bytes read is less than 0ffffh bytes. Since 0ffffh is the maximum bytes to read into the buffer at any one time, if the result of the read is less than that, then the end of the file has been reached

- Note: You only want to write out the same number of bytes you read. If you write out (or encrypt) more data than you read, your output file will be invalid

---

1.9 Requirement 9

Close both the input and output files, then, display the total bytes written to the destination file and deallocate the memory previously allocated.

1. Note: To do this, you will need to keep a running total of the total bytes read in each execution of the loop.
2. De-allocate the dynamically allocated memory original obtained in step #4

---

2 Notes

2.1 Example Ouptut

None

2.2 Hints/Tips

- I will be testing with a small file, and then a very large file. The files will not have multiples of 0x0ffffh bytes, so you will need to check correctly when copying.
- You may use the string/text print functions and console input functions only from the functions64.inc file. No other functions will be allowed.
- You could do this project in phases:
  1. Do the user input first and verify the key entry works
  2. Open the file and read part of it into a memory label, then close it to make sure that is working
  3. Open the output file and write the data read previously into it to make sure you have that working
  4. Allocate the dynamic memory and change the read and write to work with that instead of the memory label you previously created.
  5. Next create your loop which will copy each block from the input file to the output file until the number of bytes read is less than the size of your dynamically allocated memory area. When complete, the files should be identical
  6. Now create the encryption function and call it before you write the read data from the dynamically allocated memory area to the output file.

---

3 Submitting your project

- Assignment submissions will only be accept via Canvas.
- Compress/zip the following files/folders:
  - The entire 64-bit Linux application folder including all build scripts s and source files

4 Grading

Table 1: Grading Criteria

Item	**Percentage Off**
Programs do not Assemble, compile or Link	-100%
For each invalid calculation	-20%
Use of any function used not specified in this assignment	-100%
Dynamic memory allocation is not used	-50%
Dynamic memory allocation is not recovered before the program ends	-30%
Program is not a 64-bit program	-100%
Late	-100%
Copied/Plagarised code	-100%

5 Help

- Post any questions you may have on the Canvas discussion board: Chapter 11 Project
- No technical questions shall be answered via email, please post them on the discussion board
- NEVER post source code on the discussion board. This will be considered plagiarism.