Lab 4: System Calls

Contents (hide)
1 Task 0: Using nasm, ld
2 Task 1: Outputing the last part of lines
3 Task 2: First step towards Stuxnet2
4 Task 2a: A restricted stuxnet2 version: printing a list of all files
5 Task 2b: Extending stuxnet2: printing a list of all regular files that are both executable and writable
6 Task 2c: Extending stuxnet2: Add the executable code of stuxnet2 to each file obtained from Task2b
7 Task 3: Patch (Bonus Task)

As usual, you should read and understand the reading material and complete task 0 before attending the lab. To be eligible for a full grade, you must complete at least tasks 1 and 2a during the lab. Tasks 2b, 2c and 3 may be done in a make-up lab if you run out of time.

Reading Material

This lab is to be done solo (not in pairs!).

For the entire lab, do not use the standard library!
Also, do not include stdio.h, stdlib.h, or any other "standard" external header files, although you can include your own header files, if any.
This also means that you cannot use any library functions like printf, fopen, fgetc, strcmp, etc.

Task 0: Using nasm, ld

Task 0 is crucial for the successful completion of this lab! make sure you finish it and understand it before your lab session.

We will build a program which prints its arguments to standard output without using the standard c library.

1. Download start.s, main.c, util.c, and util.h.
2. Compile and link them without using stdlib (the C standard library) as follows:
   - Assemble the glue code:

     ```
     nasm -f elf start.s -o start.o
     ```

   - Compile the main.c and util.c files into an object code files:

     ```
     gcc -m32 -Wall -c -nostdlib -fno-stack-protector util.c -o util.o
     gcc -m32 -Wall -c -nostdlib -fno-stack-protector main.c -o main.o
     ```

   - Link everything together:

     ```
     ld -melf_i386 start.o main.o util.o -o task0
     ```

3. Run the program several times, each with a different number of arguments, and observe the results.
4. Look at the source code of the files, and **make sure you understand it**.
5. Write a makefile to perform the compilation steps automatically.
6. Write a new main.c that prints "hello world", or some other message of your choice, to standard output, again **not** using stdlib, using the scheme explained above, and test it.

Explanation

The file "start.s" has two purposes:

1. Each executable must have an entry point - the position in the code where execution starts. By default, the linker sets this entry point to be a library supplied code or function that begins at _start. This code is responsible for initializing the program. After initialization, this code passes control to the main() function. Since we are not using any standard libraries, we must supply the linker with _start of our own - which is defined in start.s.
2. The assembly-language source code in start.s also contains the system_call function, which is used to get a direct system call without requiring you to write in assembly language.

Note that you can link files written in different languages: an object file is an object file, no matter where it came from. All is machine code at some point!

Task 1: Outputing the last part of lines

In this task we will implement the `nothead` program:

SYNOPSIS

nothead [-n NUMBER] SOURCE DEST

DESCRIPTION
Print the SOURCE file to the DEST file, while omitting the first 10 characters in each line.

-n NUMBER, omit first NUMBER of characters instead of first 10.

In case of any error, the program should terminate with exit code 0x55 by calling the system call **sys_exit [1]**.

Examples

If the file `test` contains

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Neo: Why do my eyes hurt?
Morpheus: You've never used them before.
```

After the command `nothead test output`, `output` will contain:

```
KLMNOPQRSTUVWXYZ
o my eyes hurt?
You've never used them before.
```

After the command `nothead -n 7 test output`, `output` will contain:

```
HIJKLMNOPQRSTUVWXYZ
y do my eyes hurt?
s: You've never used them before.
```

Implementation steps

Implement the program using the following steps. Test each step before continuing to the next step!

1. Read the file characters from the file and write them to standard output.
2. Implement the omission of first characters
3. Write the result to the output file instead of stdout

Comments and tips

1. This program should use the system_call function from start.s and should not use any stdlib functions (like printf, fopen, etc.).
2. Make sure you compile and link in the same way as in task0. Feel free to adapt the makefile you wrote in task 0.
3. System calls you use here are: **sys_open [5], sys_read [3], sys_write [4], sys_close [6], sys_exit [1]**
4. Note that a "line" is anything that ends with the newline character `'\n'`.
5. You can check the exit code of your program by running `echo $?` in the shell immediately after your program finishes executing.
6. It is highly advisable **for this lab task** to read and write one character at a time (as opposed to using a larger buffer), this way the program is much simpler. However, note that this implementation is **inefficient** and should not be used in production code due to a large number of system calls that could have been minimized by using a buffer.

Task 2: First step towards Stuxnet2

Many viruses attach themselves to executable files that may be part of a legitimate programs. If a user attempts to launch an infected program, the virus code is executed before the infected program code. You goal is to write a program that attaches its code to all the executable files in the current direcotry.

In this task we will implement the `stuxnet2` program:

SYNOPSIS
stuxnet2 [-e | -a]

DESCRIPTION
Print a comment line of your choice (such as "Stuxnet 2 strikes!"), and then list of all the names of the files in the current directory.

-e , Print a list of all the names of the regular files in the current directory, which are both executable and writable.
-a , Add the executable code of stuxnet2 at the end of each regular file in the current directory, which is both executable and writable.

Note: for your convenience, we have created an header file with the appropriate structures that are used with the system calls in this task: **task2.h** .

Task 2a: A restricted stuxnet2 version: printing a list of all files

A restricted version of stuxnet2 will be implemented, as follows:
Print a list of all the names of the files in the current directory.

Some Guidelines

1. Your program should use the **sys_getdents [141]** system call.
2. Please note that the first argument for getdents is a file descriptor - it should be for the file `"."` that represents the current directory, open for reading.
3. In case of any error, the program should terminate with exit code 0x55.
4. To make things easier, you may assume that the entire directory data (returned by the getdents call) is smaller than 8192 bytes.

Task 2b: Extending stuxnet2: printing a list of all regular files that are both executable and writable

(This task and the following task may be done in a completion lab **if you run out of time** during the regular lab.

Extend `stuxnet2`, which is implemented in Task 2a in the following way:
When the flag `-e` is supplied, it will only print the names of files in the current directory, which are both executable and writable by the **owner** of the files, i.e. have execute and write permission for "user".

Note: Make sure your code prints only file names and not directory names (e.g., your code should **not** print `"."` and `".."`).

Your program should use the **sys_stat [106]** system call (not fstat or lstat).
You can use the following link to read about file permissions in Unix.

Task 2c: Extending stuxnet2: Add the executable code of stuxnet2 to each file obtained from Task2b

Warning: You probably want to be very sure that the mechanism for selecting files works correctly at this point, e.g. you may not want the program to operate on your C source code files, etc.

Extend `stuxnet2`, which is implemented in Task 2b in the following way:
When the flag `-a` is supplied, it will add the executable code (by which we mean here the entire contents of the executable file of this program) of stuxnet2 at the end of each regular file in the current directory, which is both executable and writable (i.e., all the files obtained in Task2b).

You can verify the success completion of your code by inspecting the contents of the executable files with `hexedit`.
After completing the task, execute one of the files that was "infected" by stuxnet2 and see what happens: does the virus really work, i.e. a) does it attach its code and infect the executable file(s)? b) Does it run when the infected is executed (if not, why not?)

Test your implementation on at least two files. You can use the following executables: shmul_max and ken.
Use the command `chmod u+wx <filename>` to give user write/execute permissions.

Task 3: Patch (Bonus Task)

Download the file used in this task pass and use the command `chmod +x pass` to enable execute permission for it.

Shmulik is a 2nd year Software Engineering student in BGU and he is just learning the c programming language. He decided to write the program `pass`, to help him remember all of his passwords. This program worked so well that he deleted the source code, but now there is a problem: his gmail account was somehow hacked by a Chinese hacker. Shmulik has quickly changed his password, but he needs **you** to help him update his password program before he forgets it. Of course, Shmulik does not want to actually reveal the new password to you, so you will have to write for him a program that can change the gmail password in `pass` by patching the file.

In this task we will implement the `patch` program:

SYNOPSIS
    patch NEW_PASS

DESCRIPTION
    Create a new file *pass2*, which is identical to *pass* except it prints NEW_PASS as the password for gmail.

In case of any error, the program should terminate with exit code 0x55.
In addition to the system calls used in task 1, you may want to use the **sys_lseek [19]** and **sys_chmod [15]** system calls.
Once again, remember not to use any standard library functions, only the system_call function provided.