Problem 1 -- Assemble your own cat

In this problem, you will be writing a pure assembly language program that makes system calls, and examining its operation using the `strace` utility program, which allows you to run a program with system call tracing enabled. Thus you can see the system calls that are being made along with their return values, and other events such as signal delivery and handling. strace can also be used to attach tracing to an already-running process. Please read the man page for strace.

The assembly language program is very simple: replicate your week 1 assignment, except to further simplify the coding, you will "cat" from fd 0 to fd 1, using a buffer size of 4096, and you do not need to check for write errors or partial writes. The exit code of your program should be 0, unless read fails with an error, in which case return -1.

Therefore, your assembly language program requires calling the read, write, and exit system calls, and coding a simple loop which terminates when the read system call returns a number <=0

Assembly only!: You will not be using a drop of C code, nor linking with the standard library. You will write an assembly program, which is traditionally represented with the `.S` suffix. Assemble it to a `.o` file using the system assembler `as`, and make it into an absolute executable `a.out` file using `ld`. **Repeat: do not use cc!** Your `a.out` file will be only a few hundred bytes long, most of which is the a.out header. The `objdump` or `readelf` commands will allow you to explore this and provide hours of fun, for example `objdump -d a.out` will disassemble the binary a.out file and show you the opcodes inside. (no need to attach output for these commands).

The lecture notes explain the API for both 32-bit (using INT $0x80) and 64-bit (using SYSCALL). Be mindful of which API you are running under. For 32-bit, use the flag `--32` to as and `-m elf_i386` to ld. For 64-bit, use `--64` for as and `-m elf_x86_64` for ld. Be careful: by default `as` writes its output to the file `a.out` which is not what you usually want. Use the `-o` option to override. Note that an executable which has been flagged as 32-bit architecture by ld will be run by the kernel in 32-bit mode, even if your system is natively a 64-bit system. Since the APIs are incompatible, if you have written code for the 64-bit API but assembled/linked as 32-bit, your program will be garbage and will not run. Conversely, a 64-bit program can not be run at all if you are natively running in 32-bit mode.

The system header files `/usr/include/asm/unistd_32.h` and `/usr/include/asm/unistd_64.h` contain the system call numbers for each API. Or, you can "google" this information.

Note to MacOS users: I have not evaluated this assignment on the Mac platform. However, in the past, students were able to do a similar assignment. The API is similar to Linux but some additional/different steps are required.

My lecture notes use the so-called "UNIX" or "AT&T" or "gas" assembly syntax, because that's what the Linux kernel uses. Some internet sources might be coded using the "nasm" or "Intel" syntax, which is upside down and backward compared to "gas" syntax. You will probably want to research the following topics and I'll give you most of the answer up front: Use the `cmpl` opcode to compare, followed by a conditional branch opcode (`jle` is "jump if less than or equal") to implement the loop. You will need to use the `.comm` assembly language pseudo-opcode to declare a BSS variable. For example `.comm buffer,4096,4` declares a 4096 byte buffer (the number 4 refers to the alignment requirements). The expression `$buf` then gives the address of this buffer.

Note that the first opcode of your `.text` section will be the default start-of-execution address (unless you use additional flags to `ld`) so make sure it is what you want as the first opcode!

SUBMISSION: Attach screenshots showing :

1)your assemble/link build process.

2) the strace output from running this program (test it either by typing a few lines at the terminal, or redirecting input from a small file)

3) the `$?` exit code of the program

Here is a sample of my running my code (32 bit mode)

```
$ strace ./asmcat
execve("./a.out", ["./a.out"], [/* 36 vars */]) = 0
strace: [ Process PID=6469 runs in 32 bit mode. ]
read(0, This is a test
"This is a test\n", 4096)       = 15
write(1, "This is a test\n", 15This is a test
)        = 15
read(0, of the assembled cat
"of the assembled cat\n", 4096) = 21
write(1, "of the assembled cat\n", 21of the assembled cat
)  = 21
read(0, "", 4096)                       = 0
exit(0)                                 = ?
+++ exited with 0 +++

$ echo $?
0
```