CS334: Operating Systems
Spring 2024. Assignment 0
Due: Sunday, January 28th 2024 NLT 11:59 PM
Individual, 100 pts possible

The is your C programming primer. It has to compile and run on UP's Linux VDI.

Goal:

The aim of this assignment is to illustrate how would a programmer interface with an operating system module (object) by including the module's header and linking the compiled object when making the executable program. The second aim is to review and practice the fundamental concepts from C: pointers, memory management, structs, function calls, make utility etc.

Background:

Most modern operating systems are made out of many compiled modules that together 'run' the computer hardware and provide services to the user(s). In Windows, Linux and Mac, these modules are the compiled methods that are installed with the operating system and the files are likely to have a `.dll` or `.o` file name extension.

Not only these compiled modules make up the operating system, but as a programmer you should use them to build your application – it is less work, more robust solution, optimized implementation... To do this, when you are coding your project, first you have to include the header file(s) of the modules you are intending to use in your code. The modules are nothing other than compiled functions without `main` – you did this in 305 with `gcc -c my_function.c` which created `my_function.o` module. Included header file will inform the compiler that the implementation of the module will be available during the program linking, but for now, all the compiler needs to know is the function signature. Second, when the compiler's linker combines all compiled pieces of your code (the .o objects) into the final executable program, you will have to supply the compiled module which has the actual implementation of the methods used in your code. The `make` utility will do this job. If your project folder does not have the object file, you can supply a link to where the libraries are located, or the default system libraries will be searched and used. (For more details, please review Dr. Vegdahl's 305 or 376 video lecture on compilation and linking).

All this time you have been using some of these libraries without knowing it: some libraries were used by your C compiler `(stdlib.h stdio.h)`, some are part of the OS `(sys/time.h, sys/random.h, sys/.....` many many more), and some libraries will even allow you to call methods implemented in the OS's kernel (`sys/syscall.h`). There are very few reasons why you should implement your own version of quicksort or heap or hash, as the operating system already has it (and uses it) and so should you.

The assignment:

Your job is to write a super simple C driver that takes advantage of a compiled method provided for you along with its header file (`resize.h` and `resize.o`). To keep everything super simple, both driver that you will write (cca 30 loc) and the compiled method are super super simple. The driver should illustrate a representation of a poly-line (or an edge) that is made of multiple points (`struct point`). Each point is defined by its x/y coordinates and its color. A poly-line is nothing other than a dynamically allocated array of these `point struct` pointers. As each poly-line can be of different length (size) that is not known apriori, this means that array of points and each point has to be dynamically allocated on the heap.

The compiled method `resize.o` has one function called `resize.` The function takes two parameters: a triple pointer to an array of `struct` pointers where each array slot points to a `struct` object called `point`, and the second parameter is an integer pointer that holds the array size passed to the function as the first parameter (see `resize.h`). The header file also has the `struct point` definition. The `resize` function will automatically resize the array of structs and will initialize each array slot pointer with a new `point struct`. The structs pointed to by the new (resizes) array are initialized with the rotated values of the original array struct values passed in as a parameter (x=y and y=x coords, color alternates between 0 and 1). The address of the array pointer parameter is updated with the resized array address, and the size parameter is also updates with the new (resized) array size.

The `resize` functionality, as described above, is a common function in the graphics libraries. The function would be called, if the programmer would need to upscale or downscale the resolution of a polygon line by converting the line to have more or fewer points.

Your job is to write a drive that will:

1. The program called driver.c will take one CLI argument that is a positive integer that defines the starting size of the poly-line represented as an array. The parameter defines how many line segments the starting poly-line has: `<input size>-1` line segments to be precise. Check if correct number of arguments is passed to the program, read and convert the CLI input to an int, o.w. terminate on error.
2. Dynamically (on the heap) allocate an array of `point` struct pointers with the size passed in to CLI. The array should be of `point**` type.
3. Initialize each array slot with a new instance of a dynamically allocated `point struct` (on the heap). Initialize the x, y coordinates of each point to x = size -1 and y = i (where i is the array offset of the current point). Color is initialized to 0;
4. Print each point of the poly-line on a new line.
5. Call resize with the poly-line and the size.
6. Print the new (automatically resized) array returned from the resize function.
7. Deallocate the array by deallocating each `point struct` and then the array.

Syntax: `./driver <input size>`
Example: `./driver 5`
Output: `$ ./driver 10`

```
Orig Pts: x:0, y:10, c:0
Orig Pts: x:1, y:9, c:0
Orig Pts: x:2, y:8, c:0
Orig Pts: x:3, y:7, c:0
Orig Pts: x:4, y:6, c:0
Orig Pts: x:5, y:5, c:0
Orig Pts: x:6, y:4, c:0
Orig Pts: x:7, y:3, c:0
Orig Pts: x:8, y:2, c:0
Orig Pts: x:9, y:1, c:0
poly-line resize
New Pts: x:10, y:0, c:0
New Pts: x:9, y:1, c:1
New Pts: x:8, y:2, c:0
New Pts: x:7, y:3, c:1
New Pts: x:6, y:4, c:0
```

```
New Pts: x:5, y:5, c:1
New Pts: x:4, y:6, c:0
$ ./driver 2
Orig Pts: x:0, y:2, c:0
Orig Pts: x:1, y:1, c:0
poly-line resize
New Pts: x:2, y:0, c:0
New Pts: x:1, y:1, c:1
New Pts: x:2, y:0, c:0
New Pts: x:1, y:1, c:1
```

Notes:

1. The compiled method `resize.o` was built on Debian x86 64-bit architecture. This means that you are given the object fine in machine code with is not readable to the programmer. You should link it to your program on UP's Linux VDI (which is the same architecture). It might/might not work on MacOS, it will not run on Windows.

What to turn in:

An archive (.zip) with:

1. The source code file: driver.c Your driver file must compile and run on UP's EGR Linux VM. It is your responsibility to check for compatibility
2. makefile file to compile the code and produce two targets: driver and driver.o
3. A .pdf of the write-up section in this document
4. Please don't include any executable(s) or the .o objects

Write up:

1. Did you complete all required sections and does your solution have all required functionality stated in this assignment. If not, state what functionality you successfully implemented and what is missing.
2. Draw a picture of what the array of points looks like (as a datastruture). Please initialized and show the value of each variable in each of the `point structs`.
3. Run your code through `valgrind`, copy and paste the very end of the `valgrind` output that shows the memory allocation/frees/leaks (your program should be free of memory leaks).
4. If your program would include `/usr/include/math.h` header file:
    1. What functionality does this module implement? (use `cat` or `man` commands)
    2. Copy and paste 5 sample functions/methods defined in this module? (function signature only)
    3. Using CLI, how do you find the content of the `math.h` file?
5. Shortly describe the function of the libraries with the headers included in the `/usr/include/linux` folder.

After completing this assignment, you should be able to answer the following questions (Not a part of this assignment. These questions are designed to make sure you have thorough understanding of the topics):

1. How do you, as a programmer, interact with the OS through its libraries.
2. If your program needs common functionality, which modules would you search for that implement this functionality. Examples of common functions: random number generator, socket programming, system calls, reading file system's files and directories, getting and using time, sorting, making a thread, sharing memory, making a new process,…