CMPUT 379 Assignment 1, Fall 2014

Due Wednesday, Oct 01

Note: Submit your work early. Late submission will NOT be accepted
Not knowing how to submit your work, or not being able to access eClass is not a valid reason for an extension.

Worth 20 marks (however, 5% of total course weight)

(Memory layout of programs in execution, signals)

Your goal

You are asked to program a C function with exactly the following synopsis:

```
struct memchunk {

    void *start;

    unsigned long length;

    int RW;

};



int get_mem_layout (struct memchunk *chunk_list, int size);
```

When called, the function will scan the entire memory area of the calling process and return in `chunk_list` the list of all the different chunks comprising the process's address space. By a *chunk*, we mean a continuous area of memory characterized by the same accessibility, which can be “read-only” (the value of RW is **zero**), or “read-write” (the value of RW is **one**). If the chunk is not accessible, you should set RW = -1.

In essence, your function should try to reference all locations in the 4GB 32 bit address space of the process and find out 1) whether there is readable memory at the respective location, 2) whether the memory (if readable) is also writable or 3) whether the memory is not accessible.

The function assumes that the output struct array, `chunk_list`, exists and its size, i.e., the number of entries, is equal to `size`. The function returns the actual number of memory chunks located in the process's address space, even if it is larger than `size`. In the latter case, only `size` number of initial chunks are stored in the array. The chunks must be stored in the **increasing order** of their addresses with `start` representing the starting address of a chunk, and `length` representing its length in bytes.

You must write a test program to test your `get_mem_layout` function. The test program should call `get_mem_layout` function (from *main*) and print out the chunks with their accessibility status.

Comments for completing the assignment

1. Note that you will have to intercept memory reference errors and respond to them intelligently. For this, you have to familiarize yourself with UNIX signals. They will be covered in the labs and I will also go over some examples in class.
2. Note that there is a grain of memory allocation (called a *page*), and, for a given page, it is enough to determine the accessibility of a single location falling in it. Thus, you don't actually have to scan **all** locations (page origins will do). Hint: `man getpagesize`
3. Scanning the 64 bit address space will may be problematic for you. It is intended that you do this assignment using the 32 bit address space. Please use the `-m32` bit compiler flag when compiling and linking your programs with `gcc`. Failure to do so will cause you problems. You are expected to compile your own code; a Makefile is recommended.

Marking and Deliverables

You must submit your code for `get_mem_layout` function and your `test` program. At this stage of your CS education, you are expected to deliver good quality code, which is easy to read and comprehend. **The quality and style of your code will be marked.**

- 15 marks – `get_mem_layout` function code in `memchunk.c`, `memchunk` structure definition and prototype of `get_mem_layout` function in `memchunk.h`, `test` program in `test.c`
- 5 marks – quality of coding, bug free program (including test code)

It is VERY IMPORTANT that your program follow the specifications exactly, as for marking your `get_mem_layout` function will be linked with our code that will call it (we don't mark this assignment using your test program) - be sure to follow the assignment specification EXACTLY. You may think it is better to change some of the types, but DO NOT do so.

Mandatory Coding Style

Consistent and readable C coding style is *very* important to spotting bugs and having your code readable by others. You **MUST** comply with following coding style:

- Proper indentation should be included
- TAB should be used for indentation
- NO line may be longer than 80 chars
- All functions Should prototyped
- All variables are declared at the start of a block

- C style comments must be used (not C++ style)

Note about testing:

I strongly encourage you to ssh into the lab machines and do your work there - do not spend too much time chasing bugs on your mac, or you may find the mac behaves quite differently from the lab machines. The mac is not POSIX compliant for signals.