Project 1

Interactive memory manipulation
ECEN 5813, Fall 2018
Prof. Kevin Gross

Groups

Students may work alone or in pairs. Pairs submit a single lab report and receive a common grade. It is expected that each student in a pair contributes equally and knows how all parts work.

Grading

- Coding style and clarity (10 points)
- Version control use and organization (10 points)
- Report (20 points)
- Validation by graders (10 points)
- Project meets assignment requirements and results (40 points)

Project plan

Project management starts with a plan. For this small project we will have a small plan. The plan will be part of your final report but you will also submit a draft as an early part of this project. The project plan include the following components.

Milestones - Identifiable points during development where a subset of the intended functionality is available for testing. Your final milestone represents project completion.

Tasks - A definable portion of the work that can be assigned to a project member for implementation or development. Obvious tasks include writing code to implement individual features of the project. Other tasks include testing, integration, debugging, documentation and other project overhead such as developing the project plan.

Effort estimates - Effort estimates are attached to tasks. Effort is usually expressed in man-hours. Dividing your project into more and smaller tasks often allows you to produce better estimates. Other techniques for improving the accuracy of your estimates include, collecting multiple estimates from different developers for the same tasks, expressing estimates as a range or collecting estimates in min-likely-max likely form.

Schedule - Based on your effort estimates, manpower available, work schedule and task effort estimates, calculate estimated completion dates for important milestones.

Build environment setup

This project uses a simple Gnu development environment on your Linux (virtual) machine. If you have not already, install the gcc compiler and essential libraries with the following Linux terminal commands.

```
sudo apt install gcc
sudo apt install build-essential
```

Problem statement

We're going to write an interactive command-line utility of the type you might write to help with hardware bringup on an embedded system project. The utility provides the ability to modify and examine memory and provides basic memory test functionality.

We are eventually going to use what we have created here on our FRDM development platform. But, for this project, we will not be cross compiling; We will be running on the Linux development host.

The command-line utility you write will start with a greeting to standard output and will prompt for a command from standard input. Input should accept commands and any necessary parameters for the commands. Design of this interface is up to you. You will need to document the syntax you invent as part of the project report. Consider writing the documentation first to help produce a clean design. Emphasis in your design should be on efficient user interaction and extensibility (ease of adding new commands and features in the future).

Required command functionality

Help - The initial greeting on startup must indicate how to invoke the help function. Invoking the help function will minimally list, to standard output, the available commands and how to invoke them.

Allocate memory - Uses malloc() to allocate a block of memory for use by the application. The user specifies the number of 32-bit words of memory to allocate.

Free memory - Uses free() to release the previously allocated block.

Display memory - Display contents of memory in unsigned 32-bit hexadecimal format. The user may specify an address and the number of 32-bit words to display.

Write memory - User specifies an address and the 32-bit unsigned hexadecimal value to write. Memory at the indicated location is modified accordingly.

Invert block - Use an XOR operation to invert all memory bits in a specified area of memory (address and number of 32-bit words). On completion, this command should report the amount of time taken to perform the operation.

Write pattern - Write a pseudo-random pattern in a user specified (address and number of 32-bit words) area of memory using a user specified seed value. **You may not use any library functions e.g. random() to generate this pattern.** *You may use pseudo-random number generation code you find elsewhere but you must provide attribution and you must be able to explain, in your lab report, its attributes and how it works.* On completion, this command should report the amount of time taken to perform the operation.

Verify pattern - Verify a pseudo-random pattern in a specified (address and number of 32-bit words) area of memory using a specified seed value. If pattern does not match, command should print expected and actual value and memory address where the discrepancy was found. On successful completion, this command should report the amount of time taken to perform the operation.

Functions should do bounds checking and warn the user if they attempt to read or write memory outside the allocated range (or if no memory block has been allocated) but should give users the option to ignore the warning.

Testing

A rudimentary automated test can be implemented by piping standard input from a file instead of its default connection to the keyboard. You must create and include one test case using this method that exercises all the above required functionality.

Include the test script in your code repository and include a “test” target in your makefile that invokes the test script on the program and sends output from the program to stdout.

Implementation requirements

Each command must be implemented in an independent .c file. Each command .c file must have a corresponding .h which minimally contains function prototype(s) for public functions implemented in the .c file. An additional .c file contains your main() function. Additional files beyond this are allowed if useful to the organization of your work.

A makefile is required. The default target should build the executable which, when run, satisfies the problem statement defined above. The makefile should also have a *clean* target that deletes the make product and any intermediate files generated by the build. All dependencies must be specified in the makefile such that, for instance, modification of an .h file causes the .c files which include it to be recompiled.

The project must use the Git version control system from the outset. Project files must be pushed to your team's project repository on Github.

We will be building on this work so give consideration in your development to extensibility and modularity to support modification to add additional commands, command parameters and features. You must use a structured table lookup and function pointers to dispatch commands. Dispatching commands using a large if() else if() else if()... construct is not considered an adequately extensible solution.

You may not use any library functions for your pattern generation. You may not use code from other students. Any code used from other sources must be attributed and you must document you understand how it works.

You must follow the ESE program's coding style guide.

Report questions

1. How well did your effort estimates match with actual effort? Provide examples of both inaccurate and accurate estimates and discuss any contributing factors to the success or failings of your effort estimates.
2. Were there any tasks that you forgot to include in your project plan?
3. What is the largest block of memory you can allocate? Discuss.
4. What happens if you try to read outside of the allocated block?
5. What happens if you try to write outside of the allocated block?
6. Analyze the time it takes to invert memory for different memory block sizes. Is there a linear relationship between time and size?
7. What are the limitations of your time measurements?
8. What improvements you can make to the invert function, your program or your build, to make it run faster? How much improvement did you achieve?
9. What algorithm did you choose for your pattern generator? How does it generate its pattern? What are its strengths and weaknesses?

Report

The report should be submitted at a .pdf file to the Canvas assignment associated with the project. The filename of the submission should be of the form:

ECEN5813-Project1-<last name 1>-<last name 2>.pdf

Your report should include the following sections:

- Front matter: Group members, date, project, class name
- Project plan
- Documentation for the syntax used by your command line interface
- Test case input file
- Answers to questions asked in the Questions section
- Appendix (optional): Any additional detail, sources or data that doesn't fit elsewhere in the report. This includes attributions for any code you have used from other sources including a description of how this code works.