CSE320 : System Fundamentals II Assignment 3

DUE: 7-Oct-2024

Mastering Advanced C Programming

Objective

This assignment will:

1. reinforce your understanding of advanced C programming and assembly language concepts including topics such as bit manipulation, function call, stack, union, and memory allocations.

Background

C provides many powerful features to help you perform intricate systems level work. These exercises should improve your skills with those essential features.

Instructions

You should have an Ubuntu virtual machine configured on the AWS site. If not, see me as soon as possible. Find your instance and 'start it' so it is running. Note that after each session, you should stop your virtual machine so it is not running constantly. **If you exceed a certain number of hours, you may move out of the 'free tier' and it will charge you for the usage!**

Regarding the style of this assignment, numerous partial source code files are provided. These have instructions in the comments on what to implement. Replace the lines '(Implement your code)' with lines of C code you write to perform the requested actions.

Task 1: Bit Manipulation in C

Goal: To understand techniques in C to manipulate words and bits within words.

Bit manipulation is a technique used in programming to manipulate individual bits within a binary representation of data. It's often used for tasks such as setting, clearing, or toggling specific bits to achieve certain goals. Implement the missing bit manipulation lines in the file **hw3_bits.c** given to you with this assignment.

Compile the code with your implementation and execute it. Verify that it runs correctly and produces reasonable results. Also, generate an assembly code listing using the -S switch and writing the output to hw3_bits.s. When you do this, turn off optimization (-O0). Example the Assembly and see if you can determine what sequences of assembler instructions go with which source code.

Task 2: Pointers

Goal: Learn how to access array elements using pointers.
In C, arrays and pointers are closely related, and you can use pointers to work with arrays efficiently. Implement the missing matrix multiplication lines using double pointers for 2D array matrices in the C function provided in **hw3_matmul.c**.
Please fix, add, or modify any lines if needed.
Compile the code with your implementation and execute it. Verify that it runs correctly and produces reasonable results. Again, generate assembly code (.s) from the C code using the switches -S and -O0. Study the assembly listing to learn what you can about how pointers are implemented in machine code.

Task 3: Struct, Union, and Function call by Reference

Fix or add any missing lines in the source code given in **hw3_struct_union_func.c** to exercise struct, union, and function call by reference.
Compile the code with your implementation and execute it. Verify that it runs correctly and produces reasonable results. Finally, generate assembly code (.s) from the C code as you did for the earlier tasks. Study the relationship between C source and assembly listings.

Task 4: Integration and Documentation

Do the following to integrate the code from the above 3 tasks:

1. For this exercise, each source has its own **main()**. So, to combine these, simply write a short shell script file (**execute.sh**). Use what you know of the unix command line to execute each of the separate programs. If you want, you can place a 'make' command at the top of the shell script so the one script will build all the apps and then run them.
2. Document your code in each source file thoroughly with comments explaining each section.
3. Prepare a **Makefile** to build each of the 3 applications (so include a target 'all' that has dependencies on all of the programs.) Include in the **Makefile** commands to build the assembly listings so you will need targets for files like **hw3_bits.s**, etc.
4. Prepare a brief report summarizing your efforts to code the functions that you implemented, challenges faced, and what you've learned during this assignment. Especially focus on knowledge you gained from studying the generated assembler code. Place your report in a .pdf file called **hw3_report.pdf**.

Submission Instructions

Deliverable Files: hw3_bits.c, hw3_matmul.c, hw3_struct_union_func.c, hw3_bits.s, hw3_matmul.s, hw3_struct_union_func.s, Makefile, execute.sh, and hw3_report.pdf.

Please follow this procedure for submission:

1. Place the deliverable files into a folder by themselves. The folder's name should be CSE320_HW3_<yourname>_<yourid>. So if your name is Alice Kim and your id is 12345678, the folder should be named 'CSE320_HW3_AliceKim_12345678'

2. Compress the folder and submit the zip file.
    a. On Windows, do this by clicking the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Brightspace.
    b. On Mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Brightspace.
3. Navigate to the course Brightspace site. Click **Assignments** in the top navbar menu. Look under the category 'Assignments'. Click **Assignment3.**
    a. Scroll down and under **Submit Assignment**, click the **Add a File** button**.**
    b. Click **My Computer** (first item in list).
    c. Find the zip file and drag it to the 'Upload' area of the presented dialog box.
    d. Click the **Add** button in the dialog.
    e. You may write comments in the comment box at the bottom.
    f. Click **Submit**. ← Be sure to do this so I can retrieve the submission!

Grading Criteria

The assignment will be graded for the following items:

Assignment submission follows all directives (includes required files, Makefile, etc., is backaged in a zip file…) [10% of points]

Code builds without error or warning [40% of points]

Code works and provides correct results without crashing (Incorporates proper use of C data types, pointers, arrays, structures, and dynamic memory allocation.)
[30% of points]

Code is well commented and consistently formatted (readability of the code) [10% of points]

Quality and completeness of report [10% of points]