Embedded Software Essentials

Project 1: Build System and Software Design

Zachary Vogel & James Pentz

Professor: Alexander Fosdick

February 14, 2016

Introduction

The goal of this project is to build a BeagleBone Black Build System and then write some code to test this implementation. The goal of the makefile is to be able to use the same make to build code for the beaglebone or for a regular computer. This allows us to quickly test code on a laptop, and also quickly upload it to the board. The coding problems are just for practice and something to test.

Project Procedure Results

To show that we had the output of various builds, we took some screenshots. These are shown below with appropriate captions.

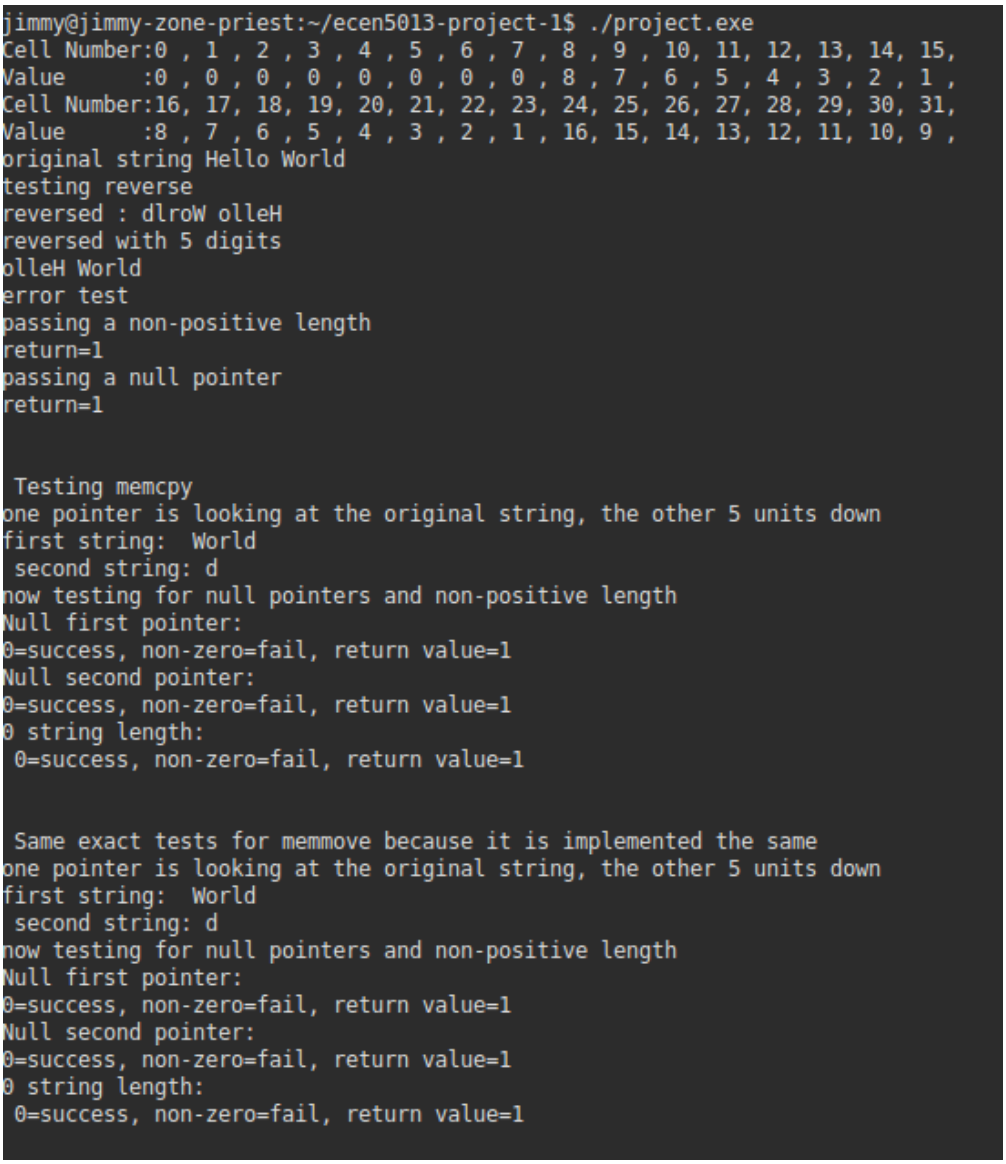

Figure 1: First a photo of the output of our code on the x86 architecture. Note that this includes the running of our test function, which is everything you see after the 4th line.

```
 memzero test
String zero'd out:0,0,0,0,0,0,0,0,0,0,0,
 Now let's test Null and zero length
Null pointer:
0=success, non-zero=fail, return value=1
0 string length:
 0=success, non-zero=fail, return value=1
That should do it
```

Figure 2: This is the rest of the output from the previous image

```
root@beaglebone:~# ./project.exe
Cell Number:0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10, 11, 12, 13, 14, 15,
Value      :0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 ,
Cell Number:16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
Value      :8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 , 16, 15, 14, 13, 12, 11, 10, 9 ,
original string Hello World
testing reverse
reversed : dlroW olleH
reversed with 5 digits
olleH World
error test
passing a non-positive length
return=1
passing a null pointer
return=1


 Testing memcpy
one pointer is looking at the original string, the other 5 units down
first string:  World
 second string: d
now testing for null pointers and non-positive length
Null first pointer:
0=success, non-zero=fail, return value=1
Null second pointer:
0=success, non-zero=fail, return value=1
0 string length:
 0=success, non-zero=fail, return value=1


 Same exact tests for memmove because it is implemented the same
one pointer is looking at the original string, the other 5 units down
first string:  World
 second string: d
now testing for null pointers and non-positive length
Null first pointer:
0=success, non-zero=fail, return value=1
Null second pointer:
0=success, non-zero=fail, return value=1
0 string length:
 0=success, non-zero=fail, return value=1
```

Figure 3: Now we see the exact same thing, but run on the beaglebone

```
 memzero test
String zero'd out:0,0,0,0,0,0,0,0,0,0,0,
 Now let's test Null and zero length
Null pointer:
0=success, non-zero=fail, return value=1
0 string length:
 0=success, non-zero=fail, return value=1
That should do it
```

Figure 4: Second part of figure 3

```
jimmy@jimmy-zone-priest:~/ecen5013-project-1$ make build
gcc -c -Wall -g -O0 -I./include -o obj/test.o src/test.c
gcc -I./include -Wl,-Map=./project.map -o ./project.exe obj/main.o obj/memory.o obj/p
roject_1.o obj/test.o
objdump -f project.exe

project.exe:     file format elf64-x86-64
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000004005b0

size ./project.exe
   text    data     bss     dec     hex filename
   4933     592       8    5533    159d ./project.exe
jimmy@jimmy-zone-priest:~/ecen5013-project-1$
```

Figure 5: The output of the make build command

```
jimmy@jimmy-zone-priest:~/ecen5013-project-1$ make upload ARCH=arm
scp project.exe root@192.168.7.2:
Debian GNU/Linux 7

BeagleBoard.org BeagleBone Debian Image 2014-04-23

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
root@192.168.7.2's password:
project.exe                                   100%   17KB  16.6KB/s   00:00
jimmy@jimmy-zone-priest:~/ecen5013-project-1$
```

Figure 6: The output of the make upload ARCH=arm command

```
[zap@WIN ecen5013-project-1]$ make compile-all
gcc -c -Wall -g -O0 -I./include -o obj/main.o src/main.c
gcc -c -Wall -g -O0 -I./include -o obj/memory.o src/memory.c
gcc -c -Wall -g -O0 -I./include -o obj/project_1.o src/project_1.c
gcc -c -Wall -g -O0 -I./include -o obj/test.o src/test.c
[zap@WIN ecen5013-project-1]$ make clean
rm -f *.exe ./obj/*.o ./asm/*.asm ./asm/*.S ./*.map ./*.out ./*.bin
```

Figure 7: The output of the make compile-all and make clean commands

As you can see requirements 1, 2, and 4-6 are completed. The testing was largely calls to make sure the functions worked, as well as testing for bad input. The main file calls two functions, project_1 and the

testing function for the memory functions memtest. We could have compiled them into seperate binaries, but decided not to. This is because it isn't a ton of output and we figured it would be better to print it all at once.

The final thing is to explain what we did for memmove and memcopy. Since the implementation details weren't that specific, our memmove is actually just a call to memcopy. memcopy itself is relatively simple, it checks for bad input and returns if it fails. Otherwise it copies everything from the first pointer into a temporary array and then copies it into the second pointer. This addresses any issues that might happen with overlapping. Another implementation would be to check for overlap first, fail if you are doing memcopy, and then copy only the overlapping region into an array. Then, move the non-overlapping part, and finally the overlapping part. This would be more efficient in terms of memory, but is slightly more intensive in terms of coding.

Conclusion

We have completed everything required of us for this assignment. This included making a very generic makefile, which is generic in the sense it works on a variety of systems. We also built the functions required for this assignment properly and tested them using the memtest function call in main.c. All components of this lab can be found on James Pentz's github at jgpentz/ecen5013-project-1. It has been a decent experience, we both gained quite a bit of knowledge on makefiles. We also gained more experience using git with 2 developers so that is nice. We were wondering if we will learn more about making makefiles that might pull code from multiple languages.