CS220 - Computer System II
Assignment 12 and 13 (200 points)

Due: 12/08/2017, 11:59pm

1 Binary Loader (100 points)

You will mimic the job of a binary loader (in a highly simplified manner). You will write a program that will load code from a file, and execute the code. You will be provided a function signature and a binary file that contains the raw bytes of the function's code. The function is a pre-compiled 32 bit binary. It is not an elf binary. It contains the raw bytes of a calculator function. It has been tested on remote.cs.binghamton.edu.

File **func.bin** contains a function with signature `int calc(char operator, int num1, int num2);`. Implement the `main` function within **loader.c**.

NOTE: The below instructions are only pointers to help you get started. You are encouraged to experiment and try different library functions. For example, you may want to experiment with `open` and `read` instead of `fopen` and `fread`. Or, if you feel adventurous, you may want to allocate memory on the heap, read the raw bytes on to the heap, call `mprotect` to make the page that contains the raw bytes executable, and execute it.

- The main function must accept exactly 4 arguments. Otherwise, main must print "Usage" message and exit. The first argument is the name of the file that contain the binary function, the second argument and fourth argument are integers, and the third argument is a character that represents an operation.
- Main function will open the file with first argument as the name. (HINT: use fopen. The second argument to `fopen` is the mode in which the file is opened. Because you are opening a binary file, the mode must be *rb*. Refer man page for `fopen` for more details).
- Main function will read the contents of the file into a local array that is large enough to contain the entire file. (HINT: use `fread`. You could either read character by character or first calculate number of bytes in the file, and read them all in one call to fread. Refer to man page for fread for usage.)
- Declare a function pointer type with name `Calc_fptr` with the same signature as calc.
- Convert the second and the fourth arguments to integers using function "atoi". Refer to the man page of atoi for usage.
- Cast the array to type `Calc_fptr`. Invoke it with the second, third and fourth arguments to main as arguments.

Write a Makefile to compile **loader.c** to generate an executable **loader**. You will get a segmentation fault when you run loader with the right arguments. This is because, you are trying to execute code that is in the local variable, which is on the stack. Stack is not executable. Now, modify the Makefile and include `-Wl,-z,execstack` flag. This is a linker flag that tells the linker to mark the stack region as executable. This is for demonstration only. DO NOT use it in production code, if you were ever to write code for a profession. Print the result, and return.

Skeleton Code

```
/* loader.c */

/* TODO: Include appropriate headers */

int main(int argc, char *argv[]){
  /* TODO: Declare an array large enough to hold the raw bytes. Raw bytes are
     best stored in byte-addressable arrays. Pick the appropriate type. Call
    it "raw_bytes"*/
  /* TODO: Declare a function pointer type that matches the calc function's
    type. Call it  "Calc_fptr" */

  FILE *fp;
  unsigned int i;
  Calc_fptr calculator;

  /* TODO if number of arguments is not 4 (5 including program name)
     print ("Usage %s <filename> <uint> <operation> <uint>\n", argv[0]) and
    exit */

  /* TODO: Open and read the binary file into raw_bytes. Use fopen and fread.
     */

  calculator = (Calc_fptr) raw_bytes;
  /* TODO: Print the result. Refer to sample input and output. */
  return 0;
}
```

Sample input and output

```
$ ./loader
Usage ./loader <filename> <int> <operation> <int>
```

```
$ ./loader func.bin 32 + 11
32 + 11 = 43
$ ./loader func.bin 32 - 11
32 - 11 = 21
$ ./loader func.bin 32 * 11
Usage ./loader <filename> <int> <operation> <int>
$ ./loader func.bin 32 \* 11
32 * 11 = 352
$ ./loader func.bin 32 / 11
32 / 11 = 2
$ ./loader func.bin 32 / 0
Floating exception
```

Input "32 * 11" is a strange one. The terminal replaces * with all the files in the directory, so you must escape it!

2 Happy Shell (100 points)

The below program implements a simple version of a shell program that runs programs that accept no arguments:

```c
/* hellosh.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

int main() {
  char line[1024];
  int pid, i;
  char *args[] = {&line, 0};

  while(1) {
    printf("Hello!!>");
    if(!fgets(line, 1023, stdin)) {
      break;
    }

    if(strcmp(line, "exit\n") == 0) break;
```

```
20      for(i = 0; i< strlen(line); i++) {
          if(line[i] == '\n') line[i] = '\0';
22      }

24      pid = fork();

26      if(pid == 0) {
          /* This is the child */
28        execvp(line, args);
          fprintf(stderr, "Hello!!: %s\n", strerror(errno));
30        exit(errno);
        } else {
32        /* This is the parent */
          wait(NULL);
34      }
    }
36  return 0;
}
```

Modify the shell program such that it can execute programs that accept arguments. For example, "ls -l" and "objdump -d -M intel foo > foo.disas" are both valid commands to hello! shell.

In your Makefile, use your modified **hellosh.c** to build an executable called **hellosh** when the command `make hellosh` is given.

3 Submission

Commit all work to your Github repository, then include the 7-digit SHA hash in your student comment for a MyCourses submission. Do not include any files!