Iowa State University
Department of Electrical and Computer Engineering
Cpr E 489: Computer Networking and Data Communication
Lab Experiment #3
Error Detection Using CRC

Objective

To write a program to generate the cyclic redundancy check (CRC) of a given data unit and to use it to detect errors.

Pre-Lab

Write the code to read data from the keyboard, and develop algorithms for CRC generation and error detection using pseudo code or a flow chart.

Lab Expectations

Work through the lab and let the TA know if you have any questions. **Demonstrate your program to the TA after you have completed it.** After the lab, write up a lab report with your partner. Be sure to

1) summarize what you learned in a few paragraphs
2) include your answers to the exercises
3) specify the effort levels of each group member (totaling to 100%)

Your lab report is due at the beginning of the next lab. Be sure to submit your **well-commented code** to the TA with your lab report for grading. Submit your report as a PDF file, and your code in .c file.

Login Information

You many login using your NetID or `489labuser`. However, it may be a good idea to use your NetID as you will be able to have the code stored in your U-drive, and you can continue working on any University Linux machine that has `gcc`.

Background

CRC (Cyclic Redundancy Check) is an algorithm to detect bit errors during data transmission. The check bits are calculated by CRC, and appended to the data to help the receiver to detect such errors. CRC is based on division. The actual input data is interpreted as one long binary bit string (dividend) which is divided by another fixed binary bit string (divisor). The remainder of this division are the check bits.

An M-bit long CRC is based on a particular polynomial of degree M, called the generator polynomial. For 16-bit CRC, the CCITT (an international telecommunication standard organization) has adopted the following CRC-16 generator polynomial: $x^{16} + x^{12} + x^5 + 1$, which correspond to the bit string of '10001000000100001'.

Division of polynomials differs from integer division. You may compute the CRC check bits for a sequence of S data bits with the following procedure:

1) First, for a given generator polynomial G of degree M, append M zero bits to S.
2) Second, divide S (which has been appended by M zeros) with G. Modulo 2 subtractions are done, and there are never any borrows. Basically, modulo 2 subtractions are the same as logical exclusive-or (XOR).
3) Third, ignore the quotient.
4) Fourth, when you get to a remainder, it is the CRC. Then, replace the appended M zeros with the CRC check bits, which may include leading zeros.

For Example:

- Input data = 11000010, meaning that there are 8 data bits.
- G = 10001000000100001, meaning this is a CCITT CRC-16 polynomial to generate 16 check bits. Therefore, 16 zero bits shall be appended to data bits to become the dividend.
- Align the leading '1' of the divisor with the leftmost '1' of the dividend and perform division, using XOR operation for each bit as follows:

```
1  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   % append 16 zero bits
1  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1
-------------------------------------------------
   1  0  0  1  0  1  0  0  0  0  1  0  0  0  0  1  0  0  0  0  0  0  0
   1  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1
   -------------------------------------------------
            1  1  1  0  0  0  1  1  0  0  0  1  1  0  0  0  0  0  0  0
            1  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1
            -------------------------------------------------
               1  1  0  1  0  0  1  1  0  0  1  1  1  0  0  1  0  0  0
               1  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1
               -------------------------------------------------
                  1  0  1  1  0  1  1  0  0  1  0  1  0  0  1  1  0  0
                  1  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  1
                  -------------------------------------------------
                        1  1  1  1  1  0  0  1  0  0  0  0  1  1  1  0   = remainder
```

The CRC value is [11111001 00001110], and thus the codeword is [11000010 11110001 00001110]. Then, the receiver will perform the division again with the same generator polynomial G, and if the remainder is 0, data is ok!

Problem Description

In this lab experiment, you are required to design and develop a program that can be used for both CRC generation and CRC error detection.

- The data unit shall accept 16 binary data bits (represented as '0' or '1' digits) from the keyboard and store them in a buffer.
- Your program shall compute the 16-bit CRC of the data unit using the CCITT CRC-16 generator polynomial: $x^{16} + x^{12} + x^5 + 1$.
- The CRC check bits (also represented as '0' or '1' digits) shall be appended to the data unit, and the resulting codeword shall be stored in a buffer in this format:

16-bit Data Unit	**16 bit CRC**

- Use the given routine, `IntroduceError.c`, with two arguments:

    - a null terminated version of the codeword above;
    - a bit error rate (BER), between 0.00001 and 1.

  The routine will randomly introduce errors in the codeword according to the given BER. Pass the BER value to the program as an argument in the command line.
- Use the CRC generation routine to check for errors in the codeword after running `IntroduceError.c`.
- Display the corrupted codeword and an indication of whether it contains errors or not.

Procedure

- Write a routine to accept '0' or '1' data from the keyboard.
- Write a routine for CRC generation and checking. It shall accept a 16-bit data unit in addition to a 16-bit CRC field. For CRC generation, the 16-bit CRC field shall be initialized to zeroes, while for CRC checking, the 16-bit CRC field shall contain the CRC.
- Write a main program to use the above routines to
    - accept '0' or '1' data from the keyboard,
    - generate the CRC,
    - display the codeword as "data unit + CRC,"
    - corrupt the codeword by applying the `IntroduceError.c` routine,
    - apply the CRC checking routine to the corrupted codeword, and
    - display the corrupted codeword and an indication of whether it contains errors or not.
- Demonstrate your program to the TA.
- Submit your code to the TA with your lab report for grading. Include a pseudo code or a flow chart of the algorithm in your lab report.

Exercises

1) Run your program six times with different BER values, increasing by powers of 10 from 0.00001 (10^{-5}) to 1 (10^{0}). Comment on your observations of codeword corruptions.
2) Run an experiment in which you:
    a. Generate a 32-bit codeword for the following 16-bit data unit: [01010101 01010101], using the CCITT CRC-16 generator polynomial;
    b. Bypass the `IntroduceError.c` routine and XOR the codeword generated above with the following error pattern: e =[00000001 00010000 00100001 00000000].
    c. Inspect the outcome of the CRC error checking procedure. Is this error detectable? Explain why.

How to Write and Run Your Program

All programs shall be written in C, under the Linux environment.

You should follow these steps:

- Edit your program using any of the editors available under UNIX (e.g., `pico`, `vi`, `emacs`, etc.)
- Compile your program using `gcc`.
- For programs using sockets, use the following command to compile:

```
gcc –o file file.c
```

  where `file.c` is your code, and `file` is the required executable file. Note that you can link to other libraries as needed, such as the math library using `–lm`. Also, please note that under other versions of UNIX besides Linux (e.g., Solaris), you need the "`-lsocket`" and "`-lnsl`" options for socket programs. (This shouldn't be an issue with the computers in Coover 2048.)
- You may run your program by typing the full path to your compiled executable

```
./file
```