Design, implement and test Error Detection Module in Simulated Network Environment

Souvik Dutta, BCSE-III (2018-2022)
(Roll : 00180501070)

Design

This assignment is implemented using C++ programming language. The programming paradigm used is Object Oriented Programming.

There are 3 principal components of the System, as shown in Fig. 1 :

1. Sender : Sends Untainted and Tainted Data through the Channel. Responsible for Data Generation and Data Mutation for testing the ErrorDetection algorithms.
2. Channel : Maintains a synchronous queue through which Data Transmission takes place. Responsible for allocation of Shared Memory and Semaphores and their subsequent deallocation.
3. Receiver : Receives the Untainted and Tainted Data through the Channel. Responsible for testing ErrorDetection Algorithms against Tainted Data.

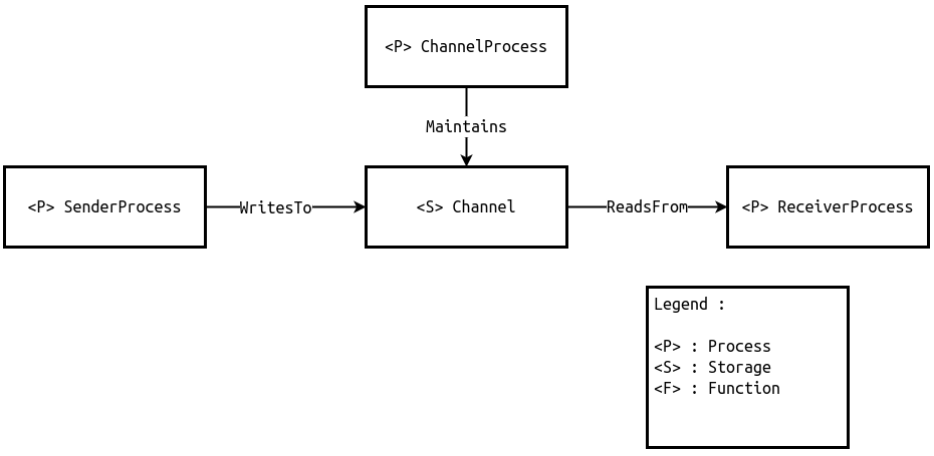

Fig. 1

Algorithm for Sender

```
Channel.open()       # Open the channel
Packet()             # Initialize a packet
                     # with random data

for()
     Packet.packFrame()        # Sets the VRC, LRC,
                               # CheckSum, CRC
     Channel.write(Packet)

     for(TAINT_TIMES)          # TAINT_TIMES : Number of
                               # To taint the Packet Data
     Packet.taintFrame()       # Randomly flip Bits
                               # (Single/Burst)
     Channel.write(Packet)
```

Algorithm for Channel

```
Channel.init()           # Allocate the Channel

for (stop == false)
     Do nothing

Channel.close()     # Destroy the Channel
```

Algorithm for Receiver

```
Channel.open()

for()
     Channel.read(UPacket)          # Receive Untainted
                                    # Packet
     for(TAINT_TIMES)
          Channel.read(TPacket)          # Receive Tainted
                                         # Packet
          ErrorsDetected = DetectError(UPacket, TPacket)
          Log.Write(ErrorsNotDetected)
```

Packet Structure

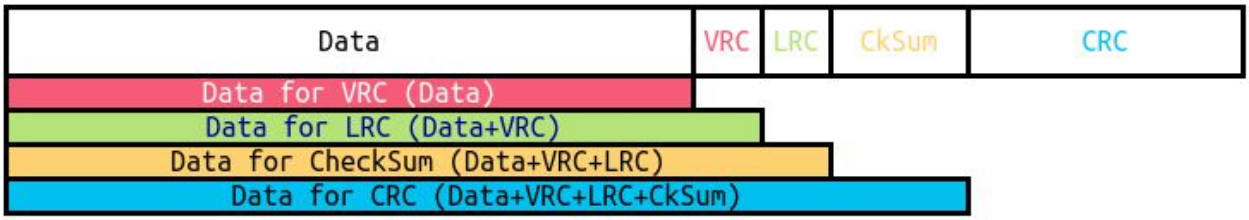

Fig. 2

SourceCode Structure

- **Header**
    - Packet.hpp - Contains Packet{} Class and PacketCompare() function.
        - Packet.Packet() - Initialises the Packet frame with random data.
        - Packet.packFrame() - Calculates CRC, LRC, VRC and CkSum on the Packet Frame. Prepares the packet for delivery over Channel.

        - Packet.taintFrame() - Introduces Bit/Burst Error (randomly choice).
        - Packet.getFrame(buffer) - Writes the Packet frame into the buffer.
        - Packet.unpackFrame(buffer) - Reads the buffer to construct the Packet frame.
        - Packet.checkError() - Returns which errors are detected for a Received Packet.
        - packetCompare(Up, Tp) - Compares a Tainted Packet(Tp) and its corresponding Untainted Packet(Up) to find which Error Detection Algorithms failed.
    - Taint.hpp - Contains Taint{} class.
        - Taint.Taint() - Initalises two helper tables for inducing bit and burst errors.
        - Taint.taintBit(buffer) - Flip a random bit in the buffer.
        - Taint.taintBurst(buffer) - Flip a series of closeby bits in a buffer.
    - VRC.hpp, LRC.hpp, CkSum.hpp, CRC.hpp - Contains VRC{}, LRC{}, CkSum{}, CRC{} classes, respectively. Here ‘ _’ stands for the corresponding VRC, LRC, CkSum, CRC.
        - .calc(buffer) - Calculate _ for the buffer.
        - .insert(buffer, _) - Inserts _ into the buffer
        - .isOk(buffer) - Checks if the buffer is corrupted or not.
        - CRC.CRC() - Initializes a helper table for fast CRC Calculation.

    - Log.hpp - Contains Log{} class.
        - write() - Shows which Algorithm failed in Error Detection.
        - endStats() - Shows summary.
    - Channel.hpp - Contains Channel{} class.
        - Channel.init() - Allocate Shared Memory and Semaphores for IPC
        - Channel.open() - Open the already allocated Shared Memory and Semaphores.
        - Channel.read() - Read received bytes from channel.
        - Channel.write() - Write bytes into channel.
    - Error.hpp - Contains Error{} class.
        - Enables enables proper Error Handling Mechanism for debugging and likes.
    - Constant.hpp - Contains a few typedefs and shared constants.
- **Main**
    - SenderProcess.cpp - Sends untainted and tainted packets
    - ChannelProcess.cpp - Maintains underlying Shared Memory and Semaphores.
    - ReceiverProcess.cpp - Receives untainted and tainted packets for evaluation

Test Cases

The entire process is automated. Errors are injected randomly. A few examples where :

1. Error is detected by checksum but not by CRC (crc16).
   Untainted Data / Tainted Data

```
E63DD466241B23569648C5D2E33EE9F371AFD0CCE53699F688886850E6AB3F4E5138F7C61243134B2F513298F5C507594D3D8BB3556517BE3FD50087A81812AB
E63DD466241B23569648C5D2E33EE9F671AFD0CCE53699F688086850E6AB3F4C5038F7C61243134B2F513298F5C58F594D3C8B93556517BE7FD50087A81812AB
                          ^               ^          ^ ^                       ^     ^  ^        ^
```

2. Error is detected by VRC but not by CRC(crc16).
   Untainted Data / Tainted Data

```
EAFFFC009AE663421173676A8F54B4121F833288B373933394FCAE86DE77365E664E92302123CB5C849A238F2016DCB15D7FCCA026E7D7EF503B004B6A175C2E
EAFDFC009AE663421173676A0354B4121F833288B373933394FCAE86DE77365E664E92302123CB5C849A238F2017DCF15D7FCCA026E7D7EF507B004A6A175C2E
___^________________^____________________________________________________^__^________________^____^_______
```

An extensive list(Errors.txt) is attached with the document.

Result and Analysis

% is calculated by the following formula =

$$(ErrorNotDetected)/(NumPackets) * 100$$

Runs	VRC	LRC	CkSum16	CRC16
1	46.236115%	0.918449%	0.138395%	0.001511%
2	46.237466%	0.918807%	0.138468%	0.001529%
3	46.235023%	0.919800%	0.138611%	0.001428%
4	46.231511%	0.925847%	0.135049%	0.001260%
5	46.215128%	0.919305%	0.134611%	0.001659%

CRC32 showed no error over Random Mutations

- Average error percentage for VRC =46.231047%
- Average error percentage for LRC = 0.920442%
- Average error percentage for CheckSum (16 bit) = 0.138426%
- Average error percentage for CRC (16 bit) = 0.001477%

The above data shows that CRC, although computationally similar to VRC, LRC or CheckSum, is more sensitive to Erroneous Data than the other algorithms.