CS5310.251/252, Spring 2016, Programming Project

Simulation of the Ethernet

Issued: April 20, 2016
Due: Midnight of Sunday, May 8, 2016

This project is to simulate the classical Ethernet. The goal of the project is to practice basic socket API programming through a client/server application. It is a *simulation* in the sense that the Ethernet is simulated by multiple processes on multiple machines. Each station in the Ethernet is simulated by a process running on one of the workstations in the Linux lab, and the common bus is also simulated by a process.

1 The Ethernet

The IEEE 802.3 protocol, commonly known as the *Ethernet*, is a *1-persistent CSMA/CD* protocol. Stations are connected to a common communication *bus* through which communications are carried out. The basic ideas are:

- A station wishing to send a message frame will first check if the communication bus before sending out the frame.
- If the bus is busy, the station will wait. If the bus is idle, it will send out the frame.
- A collision occurs when two or more frames from different stations collide on the bus. A collision can occur either during the sending period, or the uncertain period (in which the station has sent out the frame, but is not sure if the frame has arrived safely to its destination station). A station will stop transmission immediately if it detects the collision during the sending period. It then waits for a *random period* of time before trying retransmission.
- The random period mentioned above is determined by the well-known *binary exponential backoff* (BEBO) algorithm:
  - After the first collision, the time is divided into discrete slots whose length is equal to the worst case round-trip propagation time.
  - After the first collision, a station waits for either 0, or 1 time slots before attempting its first retransmission.
  - If the first retransmission collides too, it will wait for either 0, 1, 2, or 3 time slots before the second retransmission.
  - In general, after the i^{th} collision, where $i \leq 10$, a station will wait for a random number of slot times chosen from the interval from 0 to $2^i - 1$. However, after 10 collision, the random interval is fixed between 0 and $2^{10} - 1 = 1023$ time slots.
  - After 16 collisions, the Ethernet controller for a station suspends retransmissions and reports the failure to the data link layer.

2 Simulation System Structure

We simulate an Ethernet with a collection of processes that execute on a collection of UNIX machines in the Linux lab in the Computer Center. The project language is **C**.

All interprocess communication is using UNIX *sockets*. UNIX provides several kinds of sockets and you can choose whichever sockets you like for the project. You also have the freedom to choose either TCP or UDP as the underlying transport protocol.

The Ethernet is composed of the following major processes:

(1) *Station Process* (SP). One station process for each simulated station. For flexibility the number of such station processes allowed should be a variable. But for implementation purpose you can assume in the project there are at most ten station processes, i.e. we are simulating an Ethernet consisting of at most ten stations.

(2) *Communication Bus Process* (CBP). There is only one such process, which simulates the common communication bus in an Ethernet.

For every station, the sequence of frames to be sent, together with the destination station number, is specified as simulation input data file that will be read by that particular station (cf. Section 3).

Fig.1 illustrate an Ethernet with three stations.

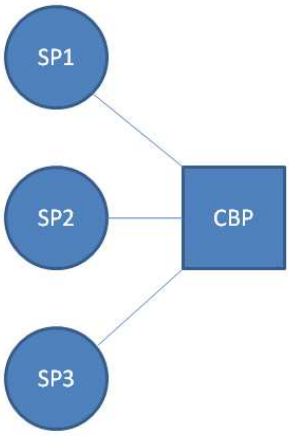

Figure 1: Illustration of the Programming Project

In this case the three stations cannot communicate to each other directly. A station, say S_1, has to send a message (intended to S_3) to the CBP first, which in turn will send the message to the destination station S_3.

2.1 The Frames

We assume that frames are of fixed size for simplicity. To simulate the existence of *sending period* each frame is sent out as two parts. Specifically, when a station sends out a

frame, it will sent two messages. These two messages together represent a single frame. In order for the CBP to be able to realize that these two messages are for a single frame, each frame has an associated *frame number*, which are attached as part of the two messages.

2.2 Communication Bus Process

The main duty of the CBP is to function as the single communication bus shared by all stations in an Ethernet. Therefore CBP must simulate collisions, accept frames from a source station and resend each frame received to its destination station if no collisions have occurred.

Every time a message is received, the CBP will inspect the message to see if this is the first part or the second part of a frame. If this is the first part of a frame, the CBP will put it into an internal buffer. If this is the second part of a frame, the CBP will fetch the first part of the corresponding frame and send it to its destination station. It will then send out the second part of the frame to the destination station.

At any time the CBP can only hold messages of a single frame from a single station (this is the nature of the communication bus). Any of the following three situations signals the occurrence of a collision:

- The first part of a new frame is received. Just before the second part of that frame is received, the first part of another frame arrives.
- Both parts of a new frame are received, and just before the first part of this frame is sent to its destination station, the first part of another fame arrives.
- Both parts of a new frame are received, and the first part of this frame has been sent to its destination station. However, just before the second part of this frame is sent out, the first part of another fame arrives.

Whenever a collision occurs, the CBP just discards the remaining parts of frames belonging to stations involved in the collision. It will then send a message to all stations involved in the collision to inform them the occurrence of a collision.

2.3 Station Processes

The operations of an SP is simple. Here is a brief summary:

(1) It first reads a line from its simulation input data file.

(2) It then sends the first message representing the first part of the frame to the CBP.

(3) If it receives a collision message from the CBP, it will act according to the BEBO algorithm. Otherwise, it will sends out the second message representing the second part of the frame. It will wait for one slot of time. If there is no collision message received during this time slot, it assumes that the previous message has been safely received and therefore it will begin to send the next frame.

2.4 Interrupts

Notice that after sending out the first part of a frame, it is possible for an SP to receive an incoming message from the CBP. Similarly, after receiving both parts of a frame but before sending out the first part, or after transferring the first part of a frame but before sending out the second part, the CBP may receive a message from another station.

In order for these processes to have this ability of being interrupted, various techniques such as *signals*, the system call *fcntl*, or *ioctl* can be used. There no specific requirement as which one should be used as long as what you use works.

2.5 Time Slots

The BEBO algorithm requires the use of the time slots. For this project, each time slot can have a duration of 100 milli-second (i.e. 1/10 of a second). However, you may change this value according to the load of the physical machines in the Linux lab during the simulation. Because the *sleep* system call provides a resolution of seconds, you may use the *select* function to have a more accurate clock. However whatever you use, the main idea of the project is the same. So even if you just use the simple *sleep* function, the basic idea of the project is still sound. But a more accurate timer implementation makes the project more interesting.

The duration of a time slot is crucial for the Ethernet and should be large enough to prevent a station from mistakenly thinking that its frame has been safely sent out.

2.6 Inter-machine Communications

The *socket* system calls are capable of providing inter-machine communications. You can use either TCP or UDP APIs to implement the project. The material presented in class and examples from the textbook should be sufficient for this project.

3 Controlling the Simulation Using Events

In order for us to be able to create different communication scenarios, we need to control the progress of the simulation. As stated before, *simulation input data files* are used for this purpose. Each SP has an associated simulation data file that will read only by that process. The simulation file is real simple. Here are several sample lines that could be in the simulation file for station 1:

Frame 1, To Station 3
Frame 2, To Station 4
Frame 3, To Station 3
Frame 4, To Station 2

4 Simulation Result

The result of the simulation is the log of activities that occur in the various stations and the CBP. An SP records the following kind of information in a separate file:

1. "Send part 1 of frame 1 to Station 3"
2. "Send part 2 of frame 1 to Station 3"
3. "A collision informed, wait for 1 time slot"
4. "Send part 1 of frame 1 to Station 3"
5. "A collision informed, wait for 3 time slot"

A CBP records the following kind of information:

1. "Receive part 1 of frame 1 from Station 1, to Station 3"
2. "Receive part 2 of frame 1 from Station 1, to Station 3"
3. "Transfer part 1 of frame 1 from Station 1, to Station 3"
4. "Receive part 1 of frame 1 from Station 2, to Station 4"
5. "Inform Station 1, Station 2, a collision"

5 Project Report

Your project report must include the following:

(1) Program files

- The program must contain a file that provides itemized description about what your have implemented and what you have not been able to complete. For each implemented feature in your project, a brief description about the method you use. For instance, how have you implemented the interrupts?
- The whole program must be well-commented. For instance, important functions must be concisely commented about the parameters needed, the main duty performed, values returned, and etc.

The whole program must be self-compilable. Be sure to include all files needed to compile your program.

(2) A README file that clearly describes how to compile and test your program.

(3) Simulation results. Please supply your input and corresponding output. Your project may be tested using your own input and may also be tested using my own input data.

The project should be submitted through the class form homework/project panel. Please follow the upload instructions.

The upload page will be closed after the due date so that I will have time to grade it. No email submission is accepted.

Each project will be tested during grading. Independence of project implementation will be rigoriously verified. The ability for the clients and server to be able to run on different computers, not just on a single computer, is important. Failing to complete essential features of the project, non-independence of project implementation, and not following the stated project submission instruction will result in low scores and will affect your final grade proportionally.