UCD School of Electrical, Electronic and Communications Engineering

EEEN20060 Communication Systems

# Assignment 1 Report

**Author:** Paul Doherty **Student Number:**  10387129

**Working with:** Colin Harper

### Declaration of Authorship

I declare that all material in this report is my own work, except where there is clear acknowledgement and appropriate reference to the work of others.

Signed: . . . . . . . . . . . . . . . . . . . . . . . . . Date: . . . . . . . . . . . . . . . .

## Introduction

The aim of this assignment was to design and implement communication protocols to allow transfer of files from one computer to another, using the serial ports.

A file is stored on the hard disk of a computer as a collection of bytes, in a particular order. Our goal, therefore, was to transfer all these bytes, in the correct sequence, to another computer, where they could be stored in another file.

This problem was broken into three parts: physical layer, link layer and application. These are discussed in detail in the sections below.

## Physical Layer

The Physical layers’ operation is to transfer bits from sender to receiver. This requires a medium, called a channel. Wires made from copper are the most common medium chosen for the channel. Another example would be fibre optic cable which uses the principle of refraction to carry signals, this medium is much faster than copper wires but also much more expensive.

In this experiment we are given no choice as to what specifications the channel should be as we have to use the serial ports provided for our computer. The physical layer protocol was constrained by the capability of the serial port hardware.

### Port Configuration

For this experiment the following port configurations were used:

* Bit rate – 4800
* Data bits in group – 8
* No parity
* One stop bit

The file can only be sent if these parameters are implemented. Otherwise, the following error message will be displayed: “error setting port parameters”

### Time Limits

The following time limits were chosen:

serialTimeLimits.ReadIntervalTimeout = 0;

serialTimeLimits.ReadTotalTimeoutConstant = 100;

serialTimeLimits.ReadTotalTimeoutMultiplier = 5;

serialTimeLimits.WriteTotalTimeoutMultiplier = 5;

serialTimeLimits.WriteTotalTimeoutConstant = 10;

### Implementation

We used the functions provided to us for the majority of this assignment. On testing our program it was found that we had to change PHY\_open function in order to support the link layer protocol. The PHY\_open function is where our chosen time limits are stored; these had to be modified to suit our link layer protocol.

## Link Layer

The role of the link layer is to provide an agreed service to the network layer using the bit transmission service of the physical layer. This involves organising the stream of bits to be sent called the frame. The frame consists of a header (start of frame), a stream of bits which is the actual information to be received (middle of frame) and the trailer (end of frame). Decisions have to be made on how the header and trailer bits will be identified and implemented. On sending data the link layer must provide a header to mark the beginning of the information to be processed and a trailer to mark the end of the incoming information. On receiving data the link layer must be able to recognise the incoming data and decide whether or not the incoming data is a good batch or a bad batch. The link layer also controls the flow of bits for example if the receiver becomes too busy then the sender must slow down.

We decided to use a Stop and Wait protocol. The basic idea here is the sender sends a frame containing a block of data then stops and waits for a reply from the receiver. When the receiver receives the block of data, it sends back an acknowledgement announcing if the data it just received is good or bad. If good then a positive acknowledgement is sent (ACK), if bad then a negative acknowledgement is sent (NACK). A problem with this is that the acknowledgment itself could contain bad data. The stop and wait protocol can be made more reliable by adding timeouts i.e. if no acknowledgment is received within the time limit then the file is sent again and enquiries i.e. sender asks receiver to send acknowledgement again.

We chose this protocol because the software can be easily created also and it’s the most relevant to our task since the experiment only requires us to send one file to one computer.

### Protocol Details

In our protocol 3 types of frames were used:

* The main frame which consists of the header, block of data and trailer.
* An Acknowledgement frame to declare if the data was transmitted correctly or not. This consisted of a positive acknowledgement (ACK and a negative acknowledgement (NACK).

The stop and wait protocol greatly reduces the risk of a bad file being stored in the receiving computer. Negative acknowledgements and timeouts are the two functions used in this protocol to tell the sender either to resend the last block of data or to continue sending new data.

When a frame is transmitted, the sender will decide if it is good or bad and send the corresponding acknowledgement frame, ACK or NACK respectively. On receiving an acknowledgement, the sender must then act appropriately by either continue sending new information or to resend the last frame of data.

The timeout function is in place so that if the event occurs where no acknowledgement is received by the sender after the given time then the sender will resend the last frame. It is very helpful in the event that the acknowledgement frame returns broken and is unreadable by the sender.

We decided to use this type of stop and wait protocol since it best suits the task at hand. Our protocol also features a loop inside the function which terminates the program if it so happens that a NACK acknowledgement is received 10 times for the same block. This flags a major problem with software.

### Frame Design

Header:

Headers would normally consist of start of frame marker, sequence number and byte count. However, since our task only involve sending a file from one computer to another ours looks like this:

|  |
| --- |
| Start of frame marker |
| Sequence number |
| Byte count |

This header will always be made up of 3 bytes, the start of header marker (represented by 1 on the ASCII table), sequence number (determined by an incrementing loop) which numbers the block being sent and the byte count (determined by a counting loop) which is the number of bytes being sent in the block. Each of these 3 sections contains 1 byte.

Block:

The block consists of the bytes that represent the actual information to be received. The bytes are got from the file that is to be transferred. A max block size has to be defined since the number of bytes sent has to be below 256, this because of our checksum function. We chose a max of 80 bytes for the block. This is so that each frame wouldn’t be too big for the computer to handle or too small so that a file would take several frames to transmit.

Trailer:

Our trailer consists of a checksum and an end of transmission.

|  |
| --- |
| Checksum |
| EOT |

Our checksum takes the sum of the block and trailer of modulus 256.

The total frame is as follows:

|  |
| --- |
| Start of frame marker |
| Sequence number |
| Byte count |
| Checksum |
| EOT |

Acknowledgement:

The acknowledgement we chose to increment consists of an ACK or NACK, a sequence number and a checksum:

|  |
| --- |
| SOH |
| Sequence number |
| Byte count |
| ACK/NACK signal |
| Checksum |
| EOT |

We used ASCII code table 6 for the ACK acknowledgement and ASCII code table 21 for the NACK acknowledgement. The checksum used here is also modulus 256 and is the sum of the 1st four bytes.

checkSum = C\_SOH+seq+type+1;

### Optimum Frame Size

With the parameters chosen above, we calculated the optimum size for our data frames.

Since our header consisted of 3 bytes, a block size of maximum 80 bytes and a trailer of 2 bytes our total frame size will be 85 bytes.

### Services

From the link layer a protocol is created for the application layer. The link layer uses the physical layer to send the frames. From the functions used in the link layer the transfer of files from one computer to another is made possible.

### Implementation

We wrote functions to carry out several of the a tasks needed for the complete program.

Buildframe:

(int buildFrame(unsigned char \*frame, unsigned char \*data, int nByte)

This function inserts the header, data and trailer bytes into the frame. This function also calculates our checksum. This checksum value is sent along with the frame and is cross-examined at the receiving end to ensure that the frame has been transferred and received correctly.

//Put header bytes into frame.

frame[0] = C\_SOH; //adding the SOH to the frame

frame[P\_SEQN] = seqNum; //adding the Sequence numbe to the frame

frame[P\_BCNT] = nByte; //adding the byte count to the frame

for(int j = 0; j < nByte;j++) //loop to write the data block to

the frame

{

frame[j+3]=data[j];

}

int checksum = 0, i;

for(i=0; i<nByte+3; i++)

{

checksum += frame[i];

}

frame[nByte+3] = checksum%256;

//end of frame byte

frame[nByte+4] = C\_EOT;

// Return the size of the frame

return HDR\_SIZE+nByte+TRL\_SIZE;

Send acknowledgement:

(int sendAck(int type, int seq)

The job of the above function is to allow the receiver to send the sender an acknowledgement regarding the information it just received. It is by using checksum that the received information is considered as good or bad and from this the appropriate acknowledgement is sent. Another function called PHY\_send is used to send the ackframe. An error will occur if the value is less than 0 and from this the value -1 will be returned. The ackframe can be considered successfully sent if a value greater than or equal to zero is returned. From this the value 0 is returned to indicate success.

ackframe[0] = C\_SOH;

ackframe[P\_SEQN] = seq;

ackframe[P\_BCNT] = 1;

ackframe[P\_TYPE] = type;

checkSum = C\_SOH+seq+type+1;

ackframe[4] = checkSum%256;

ackframe[5] = C\_EOT;

retVal = PHY\_send(ackframe, ACK\_SIZE);

if(retVal < 0)

{

return -1; // error

}

return 0; // success

Checkframe:

(checkFrame(unsigned char \*frame, int nByte)

By using checksum, this function is used to examine a received frame for errors. For the checksum to return positive, the checksum in the checkframe must equal the checksum sent. False is returned if these values are not equal.

int checksum = 0, i;

for(i=0; i<nByte-2; i++)

{

checksum += frame[i];

}

checksum = checksum%256;

if

(

frame[nByte-2]== checksum

)

// If all tests passed, return true

return true;

else{

printf("CF: size %d, calc %d, rx %d\n", nByte, checksum,

frame[nByte-2]);

return false;}

} // end of checkFrame

Getframe:

(int getFrame(unsigned char \*buffer, int maxSize, int timerNum)

Here, the receiving end is told to extract the data block out of the frame of received bits. It does this by searching the frame for the start of header marker. The PHY\_get function is called to extract one byte at a time and save it in the array buffer. Should an error be detected it will be reported and the program will terminate. When the start of header is found all the bytes are put into an array called buffer until the next start of header is found or until the cycle times out. In the situation where no useful bits are found getFrame will continue until the timer runs out and return 0.

do

{

retVal = PHY\_get(buffer, 1, PROB\_ERR); // get one byte at a

time

// If there is an error, report it and give up

if (retVal < 0) return retVal; // check for error and give up

}

while (((retVal < 1) || (buffer[0] != C\_SOH)) &&

timeOK(timerNum));

// until we get a byte which is start of frame marker, or timeout

// If we are out of time, return 0 - no useful bytes received

if (timeUp(timerNum)) return 0;

## Application Layer

The application layer program has to use the protocols of the other two layers to establish host-host connections. Our application layer runs as follows: when running, it asks the user to enter what mode it wants go into; either debug or quiet mode. Next, it asks the user if they want to send or receive a file. If the user wishes to send a file the next and final prompt is to enter the file name to be transmitted e.g. sample.txt. If the user chooses to receive a file the next and final prompt is to enter a name for the incoming file to be saved as e.g. newfile.txt. The file is saved in the same folder that our program is saved as.

### Design

We decided to transfer the file in blocks of 80 bytes. This would give a frame size of 85 bytes which would be a suitable choice for the link layer protocol.

We decided to transfer the file name sample.txt

We decide to mark the end of the file by using a byte counting while loop. This loop is designed to take in the bytes being received and saving them in a new file until the byte count is less than 80. Once a block that is less than size 80 bytes has been received the loop will be exited and the file will be saved at the end of the function.

### Implementation

We were given a simple program to do most of this, but it had no facility to transfer the name of the file, or to mark the end of the file. We modified it so that when less than 80 bytes was received the end of the file was recognised.

while ( nRx == BLK\_SIZE ); // repeat until end of file, indicated by

the last block being smaller than 80 so the loop will exit.

if (debug) printf("RX: End after %ld bytes\n", byteCount);

When sending the file name the above code was not used because our program was designed to give the user the option of entering the preferred name of the incoming file to be saved as.

## Testing

The first testing we did on our software was sending a file called sample.txt. This test was unsuccessful. This was because of small mistakes in the coding e.g. forgetting to initialise certain values such as checksum. Eventually after trial and error we got the sample.txt file to send.

After the 1st test returned successfully we next attempted to send an image of the UCD crest(crest.jpg). Again, our 1st attempt failed. This was due to some coding errors in the physical layer. Certain values in the timeouts did not suit our transmission. We had to change the Read Total Timeout Constant and Read Interval Timeout. After these values were adjusted our next attempts were all successful. We changed serialTimeLimits.ReadIntervalTimeout = 20; to ReadIntervalTimeout = 0 and serialTimeLimits.ReadTotalTimeoutConstant = 10; to ReadTotalTimeoutConstant = 100. After making these changes our software worked successfully.

## Conclusion

We have succeeded in implementing a file transfer system. It has some limitations such as should the situation arise that a file whose byte count is a multiple of 80. This would mean that the application layer would not recognise the end of the file and the program would timeout. However, it meets the requirements of the assignment since we can send and receive files between two computers connected by serial ports.