Brett Chalabian

UID 004159487

Chul Hee Woo

UID 304189459

CS118 Project 2: Simple Window-based Reliable Data Transfer

**Implementation**

Packet Model

Our packets consist of a header and body. In the header there is a character which denotes the type of packet, an unsigned long which represents the total size of the file, and an unsigned int which represents the sequence number of the packet. We have 4 characters that denote the types of packets and are *#defined* as ‘ACKPACKET’, ‘RETRANSMITPACKET’, ‘SENDPACKET’, and ‘FILENOTFOUNDPACKET’. Our packet size is fixed as per the email that we received from the TA denoting that minimum window size is 1kB. We transfer packets by passing a (char \*) pointer to our packet, and we interpret packets by converting the buffer received into a (packet \*) pointer.

Window Linked-List Model

We created a sort of Linked-List Window object that identified the packet, the state of the packet, and the timer of the packet. Our Window object keeps track of the window size by allowing a certain number of packets to exist in the window. The functions enumerated in sll.h describe methods to manage the window and the window elements that make up the linked list within. Many of the functions deal with basic linked list operations. The following

*setWindowElementStatus and ackWindowElement*

These functions deal with marking window elements as being acknowledged so the elements can later be removed if they are at the front of the window. Window elements cannot be removed immediately due to the sliding window of the Selective-Repeat Protocol. This is used in the Data Transfer part of our code to mark window elements given an incoming ACK packet’s sequence number.

*addWindowElement*

If the window size permits, the function adds a packet to a new window element and places the window element at the tail of the list. While the function returns true, packets can continue to be added to the window so the window always stays full unless all the packets have been added to the window.

*getElementFromWindow*

Returns the first element in the window that needs to be sent. Elements marked for retransmission or elements marked as not sent are returned, and their statuses and timers are updated by the main function after packets in window elements are sent to the receiver.

*cleanWindow*

Removes items from the beginning of the window if they have been acknowledged and “slides the window”. This function also manages timers: it re-marks packets as needing to be retransmitted if their timers are up.

Data Transfer

*Sender*

The sender begins by waiting for a file request packet that is neither lost nor corrupted. If the packet’s body has an incorrect filename, it sends back a FILENOTFOUND packet and waits for more packets. If the file is found, it allocates an array of packets large enough to hold the file, generates a window, and fills the window with packets. The sender then attempts to send all packets available for sending, sets their status, and then sets their timer. The sender then tries to receive all available packets from the receiver in a non-blocking way (along with checking if packets are lost or corrupted), and marks its sent packets that have been acknowledged. It then runs *cleanWindow*, and then tries to add more packets to the window if possible. It repeats until there are no more packets to send and they have all been acknowledged.

*Receiver*

The receiver sends a file request packet to the sender and waits for the sender’s response. If it receives a packet and the packet isn’t lost or corrupted, it checks the type of the packet. If the packet type is FILENOTFOUNDPACKET, it prints an error message and exits. Or else, it initializes data structures necessary to hold the file and starts placing packets in the correct position of the data structure. It then sends back an ACKPACKET to the sender with the sequence number of the file that it received. The receiver then checks to make sure all of the packets have been received and all acks have been sent, concatenates all the packet data, and writes it to a file “test.txt”.

**Difficulties Faced**

Window Implementation

We faced problems coming up with a concise way to manage the window. Our initial implementation was trying to manage the first and last packets we sent, but this did not easily capture the timer or sequence number of the packet. We developed our linked list approach to manage packets because it kept track of individual packets and was extensible to having a timer, having a variable window size, and kept track of if the packet needed to be transmitted.

Timeout Implementation

Developing a method for a timeout was fairly difficult, and initially our code was convoluted and only could check for a timeout in the first element of the list. Additionally we were kind of confused by the time() function and we thought that it returned milliseconds in a float format rather than in an unsigned integer which caused a lot of problems. We fixed the timeout issue by encapsulating it in the window methods in order to check each window element’s timer when we were trying to clean the window.