Today’s Agenda

- Usage of CRC in project 1
- Fragmentation for long messages
- Buffer/window
- Timeout handling
- Homework 2 discussion
Getting Started

• You can use any common CRC polynomials
• Create crc.h and crc.c. You’ll need to implement:
  ▪ char get_bit (char byte, int pos); // return a char with a value of 0 or 1 depending on whether the bit in the pos is 0 or 1
  ▪ char crc8 (char* array, int byte_len); // or crc16, crc32 etc.
  ▪ void append_crc (char* array, int array_len); // append crc remainder to the char array
  ▪ int is_corrupted (char* array, int array_len); // return 1 if a frame is corrupted, otherwise return 0
Steps for using CRC

• Sender:
  • Construct the frame (header and payload) and set the CRC field to zero.
  • Convert frame to char array by calling convert_frame_to_char()
  • Insert the CRC in the correct position by calling append_crc(). append_crc() will call crc8() to compute the crc remainder.
  • Send the resulting character array by appending to outgoing_frames_head_ptr.

• Receiver:
  • Check for corruption by calling is_corrupted() function on the character array.
  • If not corrupted, then convert the char array to frame and access the frame fields.

• Please note that CRC should be included in both data frames and acknowledgment frames. Even ACKs can get corrupted!
CRC 8 Computation

// Function returns the remainder from a CRC calculation on a char* array of length byte_len
char crc8(char* array, int array_len){
    // The most significant bit of the polynomial can be discarded in the computation, because:
    // (1) it is always 1
    // (2) it aligns with the next '1' of the dividend; the XOR result for this bit is always 0
    char poly = 0x07;  // 00000111
    char crc = array[0];
    int i, j;
    for(i = 1; i < array_len; i++){  // Start at most significant bit of next byte and work our way down
        char next_byte = i-th byte of array;
        for(j = 7; j >= 0; j--){  // Start at most significant bit of next byte and work our way down
            if(crc’s most significant bit is 0){
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j);  // get_bit(next_byte, j) returns the a bit in position j from next_byte
            } else{  // crc’s most significant bit is 1
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j);
                crc = crc XOR poly;
            }
        }
    }
    return crc;
}
Bitwise operators in C

• X AND Y -> X&Y
• X OR Y -> X|Y
• X XOR Y -> X^Y
• NOT X -> ~X
• Shift X by Y bits to the left -> X<<Y
• Shift X by Y bits to the right -> X>>Y

• Bitwise operators have lower priority than comparison operators
  ▪ E.g. to see whether the most significant bit of a byte is 0
  ▪ Wrong: if (byte & 0x80==0)
  ▪ Correct: if ((byte & 0x80)==0)
Fragmentation

• One way to implement fragmentation is to use the sender’s input command list

• sender->input_cmd_list_head is a doubly linked list whose nodes are of type struct cmd (in common.h)

• Before popping off the head node from sender->input_cmd_list_head check if the message > payload_size.

• If yes, then split the head node into multiple nodes where each node contains a part of the message of length <= payload_size
void ll_split_head_if_necessary(LLnode **head_ptr, size_t cut_size){
    // TODO: check if head is NULL
    LLnode* head = *head_ptr;
    Cmd* head_cmd = (Cmd*) head -> value;
    char* msg = head_cmd -> message;
    if(strlen(msg) < cut_size) {
        return;
    }
    else{
        size_t i;
        LLnode* curr, * next;
        Cmd* next_cmd;
        curr = head;
        for(i = cut_size; i < strlen(msg); i += cut_size) {
            // TODO: malloc next, next_cmd
            char* cmd_msg = (char*) malloc((cut_size + 1) * sizeof(char)); // One extra byte for NULL character
            memset(cmd_msg, 0, (cut_size + 1) * sizeof(char));
            strncpy(cmd_msg, msg + i, cut_size);
            // TODO: fill the next_mcd
            // TODO: fill the next_node and add it to the linked list
        }
        msg[cut_size] = '\0';
    }
}
Sender Buffer/Window

• We need to buffer the frames while sending them out.

• The window :
  • struct sendQ_slot {
    Event timeout; /* event associated with send -timeout */
    Frame frame;
  } sendQ[SWS];

How much timeout?
Timeout is of type struct timeval (declared in sys/time.h)
struct timeval object in C

• struct timeval{
    time_t tv_sec; //seconds
    suseconds_t tv_usec; // microseconds
}

• To calculate the timeout, get the current time and add 0.1s to it
  void calculate_timeout(struct timeval * timeout){
    gettimeofday(timeout, NULL);
    timeout->tv_usec+=100000;
    If(timeout->tv_usec>=1000000){
      timeout->tv_usec-=1000000;
      timeout->tv_sec+=1;
    }
  }

• You can use time_val_usecdiff() given in util.c to compute difference between
  two timeval objects.
Thanks!