

CS 425 Program 2: Sliding Window Protocol

Due: March 4, 2007, 11:59pm

Introduction

You must implement a sender and a receiver program that transfer data using a sliding window protocol and UDP datagrams. The sender reads the data from stdin and the receiver writes them to stdout. The sliding window protocol allow the data to be sent reliably despite frames that are lost, duplicated, or reordered while keeping the “pipe” full.

Sliding Window Protocol

You must implement the sliding window protocol as described in lecture and in the textbook. The code in the textbook is useful, but overkill for this assignment (and not complete, either). Pseudo-code for the sender and receiver is provided below. The file swp.h contains the definition of a frame in this protocol:

Bytes	Description
1	Type (TYPE_DATA or TYPE_ACK)
1	Sequence number
1	Size of body if TYPE_DATA, 0 indicates 256
1	Flags (FLAG_END)
<= 256	Body

The type field indicates the type of frame, either data or ACK. The sequence number is self-explanatory. The size field is the size of the body if it is a data frame; it is always 0 for an ACK. Since it doesn't make sense to send 0 bytes of data, the value 0 is used to indicate a full body i.e. 256 bytes. The flags field is used for flags; the only one defined is FLAG_END that indicates that this data frame is the last.

Sender Description

```
sender [-w slots] [-h host] [-p port] [-t timeout]
```

-w is used to set the sliding window size to slots. The window size must be a positive integer that is <= 128 and a power of 2 (restricting it to be a power of 2 makes it easy to find the slot for a frame by computing its sequence number modulo the window size). The default is 4.

-h connects to the receiver process on the machine host. The default is “localhost.”

-p connects to the receiver process on port port. This option is required.

-t sets the timeout before resending a frame to be timeout seconds. The timeout must be a positive integer. The default is 1.

The sender reads data from stdin and sends them until EOF is encountered.

The pseudo-code for the sender is as follows. The names in parenthesis are the C library routines or system calls you should use to implement that functionality. A “full slot” is one that contains a frame that has been sent, but no ACK received. Timeouts are implemented by using an interval timer to generate a SIGALRM signal every 100ms. On every signal the signal handler Tick goes through the full slots and resends any have timed-out. Since there is a race between filling and emptying slots as part of normal processing and the signal handler running to deal with timeouts, the SIGALRM signal is blocked while manipulating the slots.

Note that this code is simplified in that the sender will block if there is nothing to read on stdin and the SIGALRM will not be handled properly. Solving this makes the sender even more complicated than it already is, so we will assume that reading from stdin will always return data until EOF (if you are interested, the solution requires either multi-threading or select). From a practical standpoint that means you should redirect the contents of a file to stdin, rather than typing it by hand.

LFS = LAR = 0

set SIGALRM handler to the Tick routine (sigaction) set interval timer to be 100ms (setitimer)

```
while there are full slots or not EOF
  // read data from stdin and fill any empty slots
  while there are empty slots and not EOF
    block SIGALRM (sigprocmask)
    LFS++
    fill frame from stdin (fread)
    set frame's timeout
    set frame's seq = LFS
    if EOF then set FLAG_END in frame
    send (send)
    unblock SIGALRM (sigprocmask)
  read ACK from socket (recv)
  block alarm signal (sigprocmask)
  if ACK inside window
    do
      LAR++
      empty slot containing frame LAR
      while LAR != ACK's seq
    unblock alarm signal
```

Tick()

```
update clock
foreach slot
  if full and timeout <= clock
    resend frame (send)
    set new timeout
```

Receiver Description

receiver [w slots] [p port]

w is used to set the sliding window size to slots. The window size must be a positive integer that is ≤ 128 and a power of 2. The default is 4.

p specifies that the receiver should bind to port port. The default is 0, meaning that the receiver lets the operating system choose an unused port. In either case the receiver should print a message to stderr indicating on which port it is listening.

The receiver receives data from the sender and writes them to stdout. Since the data are written to stdout any messages the receiver prints should be printed to stderr. Once all the data have been received from the sender and written to stdout the receiver exits.

The pseudocode for the receiver is as follows. It is significantly simpler than the sender because it doesn't have to deal with timeouts.

LFR = 0

LAF = RWS

bind to port (bind)

```

while there are full slots or not last frame
  read frame from socket (recvfrom)
  if outside window
    drop
  find proper slot based on frame seq
  if slot already contains frame
    drop
  copy frame into slot
  mark slot full
  // send cumulative ack, if necessary
  if frame seq = LFR+1
    while slot is full
      LFR++
      write body to stdout (fwrite)
      mark slot empty
      move to next slot
  send ACK for LFR (sendto)
  LAF = LFR + RWS

```

Makefile

You are to provide a working Makefile that builds both the sender and receiver simply by typing "make". Typing "make clean" should remove the sender and receiver and any intermediate object files. You are welcome to have any additional targets you find useful.

Testing

We will make binaries for the sender and receiver available so you can test your programs. You will want to verify the correct functioning of your sliding window protocol by simulating failures such as lost data frames, ACKs, etc. For example, you can simulate a lost ACKs by having the receiver sleep longer than a timeout interval between receiving frames. This will cause the sender to timeout and resend the frame. Be creative in simulating failures - as long as some frames and ACKs get through the sender should eventually succeed in sending the data to the receiver.

Turnin

For the turnin, you will need to submit all the files that make up your sender and receiver. Design and implementation will be considered, so make sure your code contains insightful comments, uses reasonable names for variables and functions, uses no hardcoded constants, etc. You may, if helpful to us in grading, turn in a README file to help us understand your code. Use the "turnin" program on lectura to turn in your assignment under the name "cs425prog2".