CS 422 (Park) **Assignment III** **Due: Oct. 7 (Thu.), 1999**

PROBLEM 1 (30 pts)

(a) We discussed the fundamental principle underlying error detection and correction when using $m \geq 0$ additional bits to represent all possible binary strings (i.e., data) of length n which was shown to depend on the Hamming distance between code vectors. Following the framework and language used in class, explain why the parity code (say even parity) used commonly in the transmission of 7-bit ASCII characters over modem lines is able to detect all 1 bit flips or errors. What are the "legal" 8-bit patterns (i.e., code vectors)? What is the Hamming distance between them? Is it *uniform* in the sense of being the same for all pairs of code vectors? Why is error correction of 1-bit flips not possible?

(b) Assume you are the eccentric leader of a faraway kingdom and you have decreed that a byte is to be made up of 9 bits. To justify (albeit lamely) this redefinition of *byte*, you point out that there are now 2 bits that can be used to detect and correct a bit flip or bit flips when transmitting ASCII characters. What requirement must the code vectors satisfy if they are to detect 2-bit flips? What is the requirement for correcting 1-bit flips? Can such a 1-bit error correcting code for the $7 + 2$ bit case, at least in theory, exist? Explain your reasoning. Can you create a code that continues to detect 1-bit flips? (Even though one may not be able to make good use of the second bit, it must not be allowed to "hurt.") What about detecting 2-bit flips?

PROBLEM 2 (20 pts)

(a) Assume you are implementing a sliding window protocol with the sender's window size (SWS) set at N frames. Suppose each frame has a header field of length ℓ bits in which the frame's sequence number is enscribed. In the case when the sender has a lot of frames to send with the count exceeding 2^ℓ necessitating a wrap-around (i.e., reuse of sequence numbers), what problem can this cause for the receiver? What condition can one impose on SWS and ℓ such that this problem does not arise? Justify your answer. How is Stop-and-Wait impacted by this?

(b) Consider a variant of the sliding window protocol which uses negative ACKs in place of positive ACKs. Sketch at a logical plane a feasible mechanism for reliable transmission based on negative ACKs. Specify both the sender and receiver behavior. Are the two schemes intrinsically different from each other? Explain.

PROBLEM 3 (30 pts)

As a continuation of Problem 4, Assignment II, extend the client/server application such that any UNIX command can be requested by the client to be executed by the server and its output returned to the client. The request format should be of the form

$$\textit{process-ID} \# \textit{command} \# \textit{argument-}1 \# \textit{argument-}2 \# \cdots \# \textit{argument-n} \$$$

where *argument-1, argument-2,* ..., *argument-n* represent command-line arguments (possibly empty), and "\$" is an additional separator symbol. Test your program by first running the server application in the background, then executing the client `client.bin` multiple times on `date`, `who am i`, `ps`, `ps -l`, `ps -l -a`, `ls`, `ls -a`, `ls -a -l -i`, `terve` (e.g., % `client.bin ps -l`).

Use the UNIX `script` facility to record the run-time session. Hand in the hardcopy output of the script. Submit your code electronically using the `turnin` utility. You must provide adequate documentation in your code. (*Remark: Make use of string processing library functions to minimize the parsing effort.*)

PROBLEM 4 (40 pts)

As a further refinement of Problem 3, modify the client such that just prior to submitting a request to the server's FIFO, the client sets an alarm with a timeout value of Tms. If the server's response arrives before the timer expires, then this represents a "normal" operation and the alarm is cancelled. On the other hand, if the timer expires before a response arrives, the request is retransmitted to the server. Extend the request packet format such that it contains a sequence number that is incremented every time a retransmit occurs:

process-ID # *seq-num* # *command* # *argument*-1 # *argument*-2 # $\cdots$ # *argument-n* \$

Modify the server such that upon receiving the client's request and parsing it, it additionally prints the packet's sequence number. The timeout value (in msec) should be given as a command-line argument to `client.bin`. Test your program by running it multiple times on `wc /homes/park/pub/bigfile` with timeout values 5ms, 10ms, 50ms, 100ms, 200ms, 500ms, 1000ms, 2000ms, 5000ms. What does the trace of the runs tell you regarding the bahavior of the system? For this particular wordcounting request, approximate, to the extent possible, the optimal timeout value by systematic trial-and-error.

Use the same submission procedure as Problem 4 (unless otherwise specified the default).