CS 536 (Park) **Assignment X** **Due: Nov. 1 (Fri.), 1996**

PROBLEM 1

(a) Prove that if n independent Poisson processes $X_1(t), X_2(t), \ldots, X_n(t)$ with rates $\lambda_1, \lambda_2, \ldots, \lambda_n$ are combined into a single process, call it $X(t)$, via linear superposition $X(t) = X_1(t) + X_2(t) + \cdots + X_n(t)$, then $X(t)$ is Poisson with rate $\lambda = \sum_{i=1}^{n} \lambda_i$.

(b) Compute the mean and variance of the Poisson distribution with rate λ. Do the same for an exponentially distributed random variable with parameter λ.

PROBLEM 2

Assume you are a network service provider operating a switch with a bottleneck link whose output buffer can hold at most $B > 0$ packets where all packets are of a fixed size, called *cells*. Assume that the queueing behavior of your output buffer can be modeled as a random walk $X(t)$, $0 \leq X(t) \leq B$, where at each time step $t \in \mathbb{Z}_+$, $X(t)$ grows by one with probability p and shrinks by one with probability $q = 1 - p$.

If you are claiming to provide "super-reliable real-time service" whereby no packet is ever dropped at your switch, and you agree to pay a hefty fine of $\$M$ if you violate your guarantee (i.e., even a single cell is dropped), then what is the probability of your having to pay up? That is, compute

$$c = \Pr\{X(t) > B, \text{ for some } t > 0\}$$

where $X(0) = A$, $0 \leq A \leq B$ (your initial buffer occupancy is A). How much should you charge for your services to break even (your nickname is "anti-capitalist") in the long run? (*Hint: First, determine what type of barriers A and B are, respectively. Noting that c is a function of A, c_A, write a recurrence for c_A which then can be solved to find the desired probability.*)

PROBLEM 3

Use the set-up of Problem 2, Assignment VIII, however, without the stop-and-wait reliable transport mechanism, to reimplement the router, sender, and receiver with the following properties. The sender is to send a stream of 300 packets (the same header format, 1kB MTU size etc. as before) using UDP to the receiver via the router. The payload portion should contain a time stamp of when the packet was sent. It is up to you to choose a convention. After sending each packet, however, the sender is to "sleep" for s *milliseconds* (implemented using `ualarm` and `pause`) before sending the next packet where s is an exponentially distributed random variable with parameter λ. The router is to maintain a FIFO queue of size 30 packets (this is buffer space allocated by the router process in user space) to which incoming UDP packets are queued. If the queue is full, then the received UDP packet is dropped. The router dequeues packets from the queue forwarding them to their destination (the receiver). After each dequeueing action, the router is to "sleep" for r milliseconds where r is an exponentially distributed random variable with parameter μ. Make sure to implement the dequeueing (`recvfrom`) and sending part (including sleeping) "concurrently;" this will entail making `recvfrom` asynchronous (this is not necessarily so; if you don't use `pause`, you can still craft a solution). Fix $\mu = 20$ such that the mean time between successive servicing of packets is $1/\mu = 50$ msec. (This comes from the fact that an exponential distribution with parameter μ has mean $1/\mu$.) Test your system by running it with $1/\lambda = 10, 20, 30, 40, 50, 60, 70$ msec. In the router, measure the queue length each time the process "wakes up" to dequeue a packet and forward it; make sure to record this into an array (do not use `printf` since it will slow down your process) which is then printed just once after the run is over. In the receiver, keep a log of the incoming packets, time stamping the instance a packet is received. At the end of the run, plot the sequence numbers and the queueing delay (obtained by subtracting the sender time stamp from the receiver time stamp) as a function of time. Also, calculate the average end-to-end queueing delay and the packet loss rate. Compare these two quantities across the seven runs. Give an interpretation of your results. You will find a function for generating exponential random variates (`exponential_dist.c`) in the course web page.