

1. Objetivos

Esta tarea persigue construir un Multiplexor tolerante a fallas y eficiente en los enlaces con control de flujo. Es una modificación del multiplexor cliente de la tarea4, de modo de soportar una ventana de envío variable. No hay ruido ni errores entre los clientes de la tarea1 y el multiplexor cliente, como siempre. Se les provee un cliente de la tarea1 modificado para poder probar el control de flujo: en comando CAT tiene una nueva versión separado en dos comandos:

```
> 1 filename
```

Pide al servidor que envíe el archivo filename, pero no lee nada hasta que le damos el segundo comando:

```
> 2
```

Entre ambos comandos no se puede ejecutar nada en ese cliente, pero cualquier otro cliente debe poder entrar y funcionar normalmente.

2. Selective Repeat con ventana variable

El multiplexor debe implementar un Selective Repeat por conexión, para recuperarse de los errores del medio. El timeout de transmisión lo pueden fijar a 0.1 segundos. Los números de secuencia son de un byte y deben volver a 0 al superar 255. La ventana de transmisión y la de recepción deben estar fijas en 20 paquetes inicialmente.

El receptor envía sus Acks modificados ahora para avisar al mismo tiempo el tamaño de su ventana de recepción, que es el número de espacios vacíos en su ventana de recepción. Si la aplicación cliente o servidor final no está leyendo sus datos, el mux ahora no debe quedarse nunca bloqueado en un `write` y debe esperar a que sea posible enviarle los datos, informando de una ventana que se va achicando al otro mux.

El enviador debe recordar el tamaño de la ventana informado por el receptor y mantener una ventana de envío de ese mismo tamaño, es decir la cantidad de paquetes enviados esperando ACK no deben ser más que ese número. Cuando la ventana anunciada es de tamaño cero, debemos detener toda transmisión. Estos valores no afectan la retransmisión de paquetes ya enviados y que están esperando ACK, los que deben seguir siendo enviados.

Como los Acks se pierden y la transmisión puede detenerse por completo, cada dos segundos debemos enviar el último Ack con el tamaño actual de la ventana en forma permanente para cada conexión si no hubo necesidad de enviar un Ack en ese tiempo.

El formato del Ack entonces cambia y le agregamos un campo extra de un byte con el tamaño de la ventana después del checksum, quedando:

A id seq cc winsize

Como siempre, mientras un paquete de datos no logra recibir su ACK respectivo debe permanecer en la ventana esperando y retransmitiéndose. La ventana de transmisión mide la distancia entre el primer paquete esperando ACK y el último transmitido al mux. Cuando esa distancia es del tamaño máximo anunciado por el receptor, no deben aceptar más datos desde esa aplicación.

Los timeouts son por cada paquete y deben retransmitir de a uno.

3. Detalles Implementación

Deben configurar ahora todos los sockets a los clientes (pero no el de multiplexión) como *non-blocking* lo que se hace con la función:

```
fcntl(sock, F_SETFL, O_NONBLOCK);
```

Después de eso, un read o un write sobre ese socket nunca quedará bloqueado esperando datos o esperando espacio para escribir los datos. Un read sobre un socket sin datos ahora retornará -1 y pone errno = EAGAIN. Esto no es un error, solo quiere decir que no hay datos (no puede retornar 0, puesto que eso sería EOF!). Un write sobre un socket lleno usualmente retorna primero un número de bytes inferior al que yo le pasé (que son los que caben) y a la próxima retorna -1 y pone errno = EAGAIN.

También cada 2 segundos deberíamos reintentar enviarle los datos a la aplicación hasta que los acepte (cuidado con no perder el orden de los bytes).

4. Ejecución y Debugging

Como siempre.