Universidad de Chile
Facultad de Cs. Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC4303 Redes

Tarea 2

Profesor: José M. Piquer

Auxiliar: Catalina Álvarez, Tomás Wolf

Objetivos

Esta tarea persigue construir el primer acercamiento a un protocolo de envío confiable de datos, utilizando como base la estructura que crearon para la tarea 1.

Deben implementar un stop-and-wait simple, sin ninguno de los adicionales que en general se encuentran en este protocolo; es decir, los timeouts serán fijos, no se consideran múltiples clientes (todos los paquetes corresponden, necesariamente, a la misma conexión), y se ignorará la etapa de conexión.

Stop-and-wait envía un paquete de datos a la vez, esperando la confirmación de llegada antes de enviar el siguiente; de esta forma, no tendrán que preocuparse, por ahora, del manejo de ventanas.

Características de la aplicación

La aplicación tiene exactamente la misma estructura que la tarea 1, ustedes simplemente deben preocuparse del control de envío de información; eso sí, hay varios puntos importantes que se deben aclarar:

- Tal como vieron con la tarea 1, ustedes deben leer información desde un socket TCP, pero, en lugar de reenviar lo que reciben tal cual, deben en envolverlo en lo que nosotros llamamos un "header". La estructura de este header la pueden ver en el archivo adjunto *Data.h*, pero la explicaremos aquí:

    1. El header es de 6 byte. El primer byte indica el tipo del paquete 'A' indica que el paquete es una confirmación (acknowledgment o ACK), 'D' indica que el paquete contiene datos. Los siguientes 5 bytes (que se leen a partir de la posición 1, recordando que en C todo funciona como arreglos de memoria) corresponden al número de secuencia del segmento (utilizado para distinguir uno de otro).

2. El número de secuencia son 5 bytes, donde cada uno codifica un número entre 0 y 9, dando como número máximo de secuencia el 99.999.

- El header va adelante de los datos, es decir, si ustedes reciben del socket TCP 1000 bytes, deben enviar por el socket UDP 1006 bytes, donde los 6 primeros bytes corresponden al header, y a partir del sexto en adelante (determinado por la variable **DHDR**) se deben leer los datos. Es importante que hagan este procedimiento correctamente, pues bwss espera datos en ese formato.

- Dado que ahora sí es importante asegurarse que todos los paquetes lleguen, deben implementar un sistema de **timeout**; para esta tarea, el tiempo de espera para la confirmación de llegada de un paquete es 1 segundo. **Después de ese tiempo ustedes deben dar el paquete por perdido y retransmitirlo**.

Para facilidad de desarrollo, `bwss` tiene incluido dos flags que se pueden poner antes de los parámetros que utilizaron para la tarea 1: **-d** indica modo debug, que les permite obtener más información de lo que está haciendo el servidor proxy; **-L** seguido por un número entre 0 y 1 les permite incluir pérdida en la conexión (0.3 indica un 30 % de probabilidad de pérdida).

Un detalle importante que deben tomar en cuenta es la posibilidad de que al reader le lleguen ACKs, que en realidad le corresponden al writer; en este caso, los ACKs se le deben hacer llegar al writer de la forma que ustedes consideren más razonable.

Características de la implementación

Ustedes deben implementar lo pedido basándose en su propia implementación de la tarea 1. Casos especiales deben conversarse directamente con los auxiliares.

La tarea puede ser hecha entre dos personas, siguiendo las reglas comentadas en clases: **una vez han trabajado con alguien, sólo tienen la posibilidad de seguir trabajando juntos o separarse**.

Cualquier duda o pregunta o reporte de bugs, dirigirse al foro de U-cursos.

Pruebas de eficiencia

Una vez terminada la tarea, se les pide probar su implementación con pérdida de 0, 10 % y 30 %.

Para probar con pérdidas, se les recomienda probar de dos formas distintas:

1. Utilicen la flag **-L** de bwss, principalmente durante desarrollo. Una ventaja de esta alternativa es que les permite desarrollar en ambientes en que no tengan permisos de superusuario (como anakena, por ejemplo).
2. **Muy recomendado, dado que así se realizará la evaluación**. Para asegurarse de que todo funcione bien utilicen "netem":

   ```
   % tc qdisc add dev lo root netem loss 20.0%
   ```

   Y tienen 20 % de pérdida. Para modificar el valor, deben usar **change** en vez de add en ese mismo comando.

Entrega

A través de U-cursos, **no se aceptan atrasos**. La evaluación de esta tarea es muy simple: el cliente debe funcionar bien con el servidor provisto.

Entreguen los fuentes, las instrucciones para compilar todo, y un LEEME.txt donde muestren los resultados que obtuvieron.