SocketTCP – Stop And Wait

Connect:

Se escoje un número SEQ aleatorio del 0 al 100 como punto de partida, y luego se le envía al otro socket el SYN con un timer de 10 segundos.

Se intenta recibir un mensaje (los receive se harán con buff size 23, considerando un aproximado de los tamaños máximos de los header y el contenido de los datos que será de largo 3 siempre). Si se cumple el timeout, simplemente la función se vuelve a llamar a sí misma. Si se recibe algo, se verifica que corresponda a un mensaje SYN+ACK. Si no lo es, se llama a la función de nuevo.

Una vez recibido el SYN+ACK correctamente, se incremente el número SEQ en dos, y se envía un ACK. Se cambia el host de la dirección sumándole uno al anterior. Aquí termina la función.

Accept:

Se intenta recibir un mensaje. Se verifica que el mensaje recibo sea tipo SYN. Si no es así, se llama a la función Accept otra vez.

Si ya tenemos el SYN, se establece el número de secuencia con el SEQ del mensaje recibido más uno, y luego se envía un mensaje SYN+ACK con el este número. Establecemos una cantidad de timeouts permitidos para esperar el ACK (más adelante veremos por qué).

Intentamos recibir un mensaje. Si nos llega un mensaje ACK con un SEQ sucesor al que tenemos, se pudo recibir correctamente la confirmación, así que creamos un nuevo socket, con el mismo IP del socket actual, pero un host incrementado en uno. Se le establece el mismo buff size, y un SEQ incrementado en uno con respecto al actual. Retornamos el socket junto con su address.

Si en cambio nos llegó un mensaje SYN, quiere decir que el emisor no recibió el SYN+ACK, por lo que volvemos a establecer el número de secuencia con el SEQ del mensaje recibido incrementado en uno (pues probablemente el emisor volvió a generar otro SEQ de partida aleatorio). Enviamos nuevamente el SYN+ACK, y gracias al while se vuelve a intentar recibir el ACK.

Si tuvimos timeout al esperar el ACK, incrementamos la cantidad de timeouts que hemos tenido en uno y volvemos a tratar de recibir. Si ocurren 6 timeouts, es muy probable que el cliente ya haya empezado el Stop and Wait, y se haya perdido el mensaje ACK. En ese caso se sale del while y se crea el nuevo socket igual como si se hubiera recibido el ACK apropiadamente. Podría ocurrir que en realidad el emisor sigue intentando enviar el SYN pero ha fallado todo este tiempo. Para disminuir a lo más mínimo posible la probabilidad de que esto ocurra, se podría incrementar la cantidad de timeouts permitidos en base a la probabilidad de pérdida (con el costo de tener más tiempo al receptor atrapado en el while).

Send:

Se da inicio al Stop and Wait. Establecemos el byte inicial como 0 y enviamos un mensaje con el SEQ seteado en el socket y el largo del mensaje que queremos enviar en los datos.

Intentamos recibir algo con un timer de 10 segundos. SI se cumple el timeout o llega un mensaje que no corresponda a lo esperado, se vuelve a llamar a la función. Si en cambio nos llega un mensaje ACK con una SEQ mayor al que tenemos en el socket y un “1” de confirmación en los datos, entonces seteamos dentro del socket el SEQ del mensaje, y entramos al while.

Al inicio establecemos el máximo byte que enviaremos del mensaje a partir del byte inicial actual, y con eso sacamos el trozo de mensaje que enviaremos. Lo enviamos con SEQ del socket y esperamos con un timer de 10 segundos. Si se cumple el timer, simplemente se volverá al inicio del while y se intentará enviar el mensaje de nuevo. Lo mismo si no llega una respuesta que corresponda.

En caso de llegar un ACK con un SEQ mayor al del socket, se actualiza el número de secuencia, se agrega el trozo de mensaje que acabamos de enviar a una variable que guarda lo que llevamos de mensaje. Si el máximo byte que establecimos anteriormente es igual al largo del mensaje, se considera que el mensaje se envió completo con éxito y se detiene el while. Si no, sumamos en 3 el byte inicial y continuamos el while.

Recv:

Tratamos de recibir un mensaje. Una vez recibido, se entra a un while para manejar la confirmación de la llegada del largo del mensaje.

Si nos llegó un mensaje que no corresponde a nada de lo que estábamos esperando, se llama a la función de nuevo. Si nos llegó un mensaje con un SEQ menor a nuestro socket, quiere decir que el emisor sigue intentado enviar el último pedazo del mensaje anterior pues no le llegó el ACK, por lo que hay que volver a enviarle dicho ACK. Luego, se llama a recv otra vez.

Si nos llega un mensaje con un SEQ igual al del socket y los datos son numéricos, quiere decir que hemos recibido el largo del mensaje. Lo que haremos es crear dos string vacios, uno para el mensaje que vamos recibiendo, y otro para mantener un respaldo del mensaje. Iremos también registrando los bytes que hemos recibido hasta el momento y lo iremos comparando con el largo del mensaje recibido (o con el buff size si este es menor al largo del mensaje). Incrementamos en uno el SEQ del socket, enviamos el ACK correspondiente y tratamos de recibir. Si lo que recibimos no tiene el mismo SEQ que nuestro socket, quiere decir que el emisor no recibió el ACK, por lo que hay que revertir el SEQ del socket y volver a llamar a recv. Si no, podemos salir y pasar al siguiente while que será para recibir el resto del mensaje.

Si nos llega un mensaje con un SEQ igual al del socket, pero lo que se encuentra en los datos no es un número, quiere decir que en realidad estamos recibiendo la continuación de un mensaje debido a que buff size es menor al largo del mensaje. Iniciamos las variables de manera similar al caso anterior, pero esta vez al largo del mensaje se le resta los bytes que ya fueron recibidos antes. Tratamos de recibir.

Ahora sí, entramos al while que corresponde a la recepción del mensaje real. Comenzamos recibiendo (a no ser que estemos entrando por primera vez al while, pues ya recibimos algo antes en el while anterior).

Si nos llega un SEQ igual al actual, guardamos en el mensaje de respaldo lo que llevamos del mensaje real. Si nos llega un SEQ menor, revertimos lo que llevamos del mensaje real con la ayuda del mensaje de respaldo, y restamos la cantidad de bytes que llevamos. Luego, sumamos al mensaje real lo que nos llegó del emisor, incrementamos nuestro SEQ, y enviamos de vuelta un ACK con el nuevo SEQ.

Sumamos la cantidad de bytes que hemos recibido. Si los bytes que hemos recibido alcanzan el largo del mensaje, saldremos del while y retornamos el mensaje recibido completo. También saldremos si es que los bytes recibidos superan al buff size, no sin antes guardar en el socket los bytes que llevamos para el siguiente recv.

En caso de haber recibido algo que no corresponde a los dos casos anteriores, simplemente se regresa al inicio del while y recibe de nuevo.

Close: Esta función sigue una lógica muy similar a la función Connect. Las principales diferencias son que si ocurren 3 timeouts al tratar de enviar el SYN, se asume que el receptor ya cerró su conexión, por lo que el emisor hará lo mismo, y la otra es que al final el último ACK siempre se enviará 3 veces antes de cerrar la conexión.

Recv_close: Al igual que Close, esta función es similar a Accept. Las dos principales diferencias son que si al inicio se recibe un mensaje que no es tipo FIN, quiere decir que el emisor sigue tratando de enviar el mensaje anterior, por lo que se le envía un ACK con el SEQ correspondiente, y volvemos a llamar a Recv_close. La otra es que si tenemos 3 timeouts al tratar de recibir el último ACK, simplemente se cierra la conexión.

CASOS BORDE:

Se detectó un caso borde. Al tratar de enviar un mensaje con un buff size menor al largo del mensaje, puede ocurrir que justo cuando comienza el segundo recv, el emisor envíe un trozo del mensaje que contenga solo números, y el segundo recv lo detectaría como el largo del mensaje. Una manera de evitar podría ser agregar como condición que el bytesSoFar del socket sea 0 dentro del if que encapsula el caso de recibir el largo del mensaje, y también que una vez se reciba el mensaje completo, el bytesSoFar vuelva a 0.

Así, si el bytesSoFar es mayor a 0 por tratarse de un segundo recv de un mismo mensaje, no podría guardar ese trozo de mensaje como el largo. Pueden haber otras formas, lo importante es encontrar la manera de que un recv comunique al siguiente recv una señal que indique que lo que va a recibir es el contenido del mensaje.