# Socket TCP – Selective Repeat

Para debugear, se utilizaron prints a lo largo del código para mostrar su flujo, indicando los SEQs que se envían, los que se reciben, y los movimientos de ventana. También hubieron prints para mostrar índices de timers y ventanas, además de otras variables, sin embargo estos no fueron incluidos en el código final.

## EXPERIMENTOS

Lamentablemente, los experimentos de mayor duración no pudieron realizarse completo, debido a que el programa ocasionalmente se detenía por ningún motivo aparente en momentos aleatorios.

### -20 bytes

Stop And Wait:
Tiempo → 30 segundos
Segmentos enviados → 11 segmentos

Selective Repeat (ventana de tamaño 1):
Tiempo → 40 segundos
Segmentos enviados → 14 segmentos

Selective Repeat (ventana de tamaño 5):
Tiempo → 5 segundos
Segmentos enviados → 8 segmentos

Selective Repeat (ventana de tamaño 10):
Tiempo → 5 segundos
Segmentos enviados → 12 segmentos

Selective Repeat (ventana de tamaño 100):
Tiempo → 5 segundos
Segmentos enviados → 10 segmentos

### -500 bytes

Stop And Wait:
Tiempo → Llegó a durar alrededor de 10 minutos antes de detenerse.

Selective Repeat (ventana de tamaño 1):
Tiempo → 401 segundos
Segmentos enviados → 206 segmentos

Selective Repeat (ventana de tamaño 5):
Tiempo → 196 segundos
Segmentos enviados → 204 segmentos

Selective Repeat (ventana de tamaño 10):
Tiempo → 126 segundos

Segmentos enviados → 205 segmentos

Selective Repeat (ventana de tamaño 100):
Tiempo → 20 segundos
Segmentos enviados → 197 segmentos

**-1000 bytes**

Selective Repeat (ventana de tamaño 10):
Tiempo → 231 segundos
Segmentos enviados → 394 segmentos

Selective Repeat (ventana de tamaño 100):
Tiempo → 45 segundos
Segmentos enviados → 414 segmentos

**-Preguntas:**

1) El tiempo con Stop & Wait fue generalmente mayor que con Selective Repeat.
2) Con una mayor ventana, el tiempo que toma enviar un mensaje baja considerablemente. Sin embargo, la cantidad de segmentos enviados no parece haber variado mucho.
3) Aumenta el tiempo que toma ser enviado, y por su puesto la cantidad de segmentos que se envían.
4) Con un tamaño de ventana mayor, se permite un mayor envío simultáneo de mensajes, mejorando la eficacia del programa.

**FUNCIONES**

Solo se describirán las funciones send y recv con Selective Repeat, las demás funciones están descritas en la entrega de la actividad anterior.

**send_using_selective_repeat:**

Se guarda el SEQ inicial. Se crea una lista con el largo del mensaje y el mensaje en sí dividido en trozos, y se crea una ventana con dicha lista. Se crean además listas que contienen los ACKs recibidos (se marca con un 1 si se recibió), otra con los números de secuencia presenten en la ventana en cada momento, y otra con los timers.

Por cada trozo de mensaje que se encuentre en la ventana inicialmente, se extrae junto con su número de secuencia, este se envía al receptor y luego se inicia su timer correspondiente. Luego, entramos al While.

Obtenemos una lista con los índices de los timers que se terminaron. Si tenemos timeouts, primero verificamos la cantidad de índices nulos que hay en la ventana (para evitar errores al tratar de operar después de donde acaba el mensaje). Luego, para cada timeout, recorremos la ventana para ver a cuál trozo de mensaje corresponde. Una vez encontrado, se obtiene el mensaje y su número de secuencia para ser enviado nuevamente, iniciando su timer nuevamente. Una vez se termina de hacer esto con todos los timeouts, se intenta recibir (la primera vez que entramos al While lógicamente no habrá timeouts, así que pasará directamente a tratar de recibir algo).

Si es que se recibe algo, se verifica que sea un ACK y que el SEQ esté dentro del rango en el que se está trabajando. Luego, se guarda el SEQ del mensaje, se calcula el índice que le corresponde en la lista de timers, y se marca con un uno en la lista de ACKs recibido.

Si el ACK que acabamos de recibir corresponde al primer elemento de la ventana, se recorre la lista de ACKs desde el índice que corresponde al mensaje recientemente recibido. Por cada elemento, si el ACK está en 1, lo desmarcamos y movemos la ventana en una posición (además se lleva un contador con la cantidad de movimientos que se han hecho). Si la cantidad de veces que movimos la ventana equivale al largo de la lista de datos, se aumenta el SEQ del socket en el tamaño de la ventana con el objetivo de evitar que haya interferencia con el envío del siguiente mensaje, y se retorna.

Si no, extraemos los datos de la última posición de la ventana (el nuevo trozo de mensaje que apareció). Si no son nulos, se obtiene su SEQ y se guarda en la lista de SEQs. Se envía el nuevo mensaje y se inicia su timer correspondiente.

**recv_using_selective_repeat:**

Se verifica primero si el mensaje está completo. Si lo está, simplemente se guarda el primer SEQ y se inicia message_length en 0. Si no, hay que obtener el SEQ en el que se quedó el último recv, y guardarlo como primer SEQ, además de restarle al largo del mensaje lo que ya se ha recibido hasta al momento.

Similar a la función send, se crea una lista que marca los mensajes recibidos, y otra que guarde los SEQs que se encuentren en la ventana en el momento.

Dentro del While, primero se intenta recibir un mensaje. Obtenemos los SEQS que estén en la ventana en ese momento también. Se verifica que los headers y datos del mensaje sean apropiados.

Si el largo del mensaje sigue guardado como 0, el SEQ del mensaje es el inicial, y además lo que se encuentra en los datos son solo números, significa que recibimos el largo del mensaje, así que lo guardamos, y se marca como recibido en la lista. Se colocan los datos en la ventana, y se envía el ACK. Luego, vamos recorriendo la lista de recibidos para ver cuántos movimientos de ventana deben realizarse. Por cada recibido marcado como uno (y hasta que lleguemos a un 0), se obtiene el SEQ inicial para calcular índices después, sumamos los bytes que llevamos, y desmarcamos en la lista de recibidos. Si es el SEQ inicial y el mensaje aún está vacío, no se agrega el trozo de mensaje a lo que llevamos de mensaje (pues corresponde al largo del mensaje). Se mueve la ventana en uno, y obtenemos el SEQ del nuevo elemento que apareció en la ventana para guardarlo en la lista de SEQs actuales. Si los bytes que llevamos superan el largo del mensaje o el buff size, hay que terminar la función. Si el largo del mensaje es mayor al buff size, marcamos el mensaje como no completo para que lo sepa el siguiente recv. Pero si el mensaje completo guardado en el socket ya cumplió con el largo del mensaje, se marca el mensaje como completo, y se aumenta el SEQ en el tamaño de ventana.

Si el SEQ del mensaje recibido no se encuentra en la lista del SEQ, quiere decir que la ventana del emisor se quedó atrás, por lo que simplemente hay que devolverle el ACK.

Si el SEQ sí está en la lista (y por lo tanto dentro de la ventana), se calcula en qué índice de la ventana debería estar para colocarlo allí, y se envía el ACK. Luego, si el recibido está marcado como 0, lo marcamos como uno, y si el SEQ corresponde al primero de la ventana, debemos recorrer la lista de recibidos para realizar los movimientos de ventana correspondientes, sumar los bytes y mensaje que

llevamos, y obtener el nuevo SEQ que aparece en la ventana (muy similar como se hace al final del caso en que se recibe el largo del mensaje). También se verifica si se cumplieron los bytes para retornar. Si en realidad el recibido ya estaba marcado como 1, no se hace nada.

**CASO BORDE**

Un error en particular que se detectó es que cuando el mensaje se debe recibir usando más de un recv por tener un buff size pequeño, puede ocurrir por ejemplo que el primer recv reciba un trozo de mensaje que en realidad tendría que recibir el segundo recv y le llega con éxito el ACK al emisor. Luego la ventana del emisor podría moverse hasta el final y dar por terminado el envío, cuando en realidad el segundo recv todavía está esperando.

Una forma en que se puede solucionar esto sería calcular el último SEQ que puede recibir el primer recv, con el objetivo de que rechace trozos de mensaje que le corresponden al siguiente recv.