PDyTR 2018

Informe 1

Integrantes:

Aparicio Natalia, Legajo 12667/7

Eusebi Cirano, Legajo 12469/2

1 - Identifique similitudes y diferencias entre los sockets en C y en Java.

Similitudes

- Poseen características y funciones suficientes para cumplir el modelo de Cliente / Servidor.
- Ambos tienen el concepto de socket de espera y socket de emisión.
- Los sockets son siempre tanto de emisión como receción (full duplex)
- Se permite elegir el protocolo (TCP/UDP)
- La lectura funciona de la misma manera. Apenas llega algún dato al receptor, se libera (el buffer/stream) para continuar con la ejecución.

Diferencias

- Java hace uso de sockets de manera de alto nivel y C, en cambio, tiene funciones y manejos primitivos de sockets.
- El acceso al socket es a traves de un Stream en Java, y a traves de buffers en C.

2 - Preguntas

Inciso a) ¿Por qué puede decirse que los ejemplos no son representativos del modelo c/s?
En los ejemplos de la práctica podemos ver como se usan los sockets para una comunicación. El client que manda un mensaje y una vez mandado, termina la comunicación y el server que espera a recibir un mensaje, lo imprime y termina también.
Un modelo Cliente/Servidor tiene ciertas caracteristicas a cumplir:

Pasos de conexión:

- Inicialización
- Envió/recepción de peticiones
- Finalización

Características del Server

- Inicialmente pasivo (ejemplo: esperando una query)
- Está escuchando (en algún puerto), listo para responder una petición por un client.
- Al recibir una petición, la procesa y devuelve una response

Características del Client

- Tiene un rol activo
- Envía peticiones al Server
- Al enviar una petición, espera la respuesta.

Inciso b), c) y d)
Nos basamos en los archivos hechos en C para realizar los incisos de este punto.

Ambos tenemos conocimiento en C por haber realizado la materia Seminario de C y por haber tenido que usarlo en otras materias con un uso similar (Sistemas paralelos).

El approach que pensamos es con buffers de tamaño variable, utilizando punteros a char. Por esto, no nos encontramos con el problema de la limitación de un arreglo fijo/estático []. Según en el estándar C 99, la cantidad mínima en bytes con los que nos empezariamos a encontrar con problemas es de: 65,535 bytes.

Inciso b)

Para compilar los archivos en la carpeta csock:
gcc clientFromFile.c -o bin/client -lm
gcc serverBigData.c -o bin/server

Para correr cada archivo:
./bin/client compiler 9000
./bin/server 9000

Leyendo la documentación de read y write en C pudimos encontrar la razón por la que lee de a partes del socket.
Ambas operaciones devuelven la cantidad leída/enviada, pudiendo ser menor al count especificado. Esto no significa que hubo algún error sino que para un mejor manejo de memoria y de la aplicacion (performance) lee hasta una cantidad máxima (variable) de datos.

Si la conexión fuese lenta, problemas de conectividad o algunos de los dos procesos fuera más lento, al intentar leer/escribir toda la cantidad deseada, podría bloquearse el proceso intentando realizar la operación o hasta "nunca" terminar. Entonces entendemos que al hacer la lectura/escritura de a batches, devuelve el control del proceso para decidir como continuar, teniendo como dato la cantidad de datos leídos/escritos.

Doc:
http://man7.org/linux/man-pages/man2/read.2.html
http://man7.org/linux/man-pages/man2/write.2.html

Inciso c)

Para la verificación de cantidad lo hacemos enviandole dos mensajes al servidor, el primero indicando la cantidad y en el segundo mensaje, la información.
Para la verificación de contenido, con la misma función definida en ambos procesos, utilizamos un checksum.

Inciso D)
Usamos como tamaño de file 3, 4 y 6 MegaBytes de prueba. No utilizamos arhivos más pequeños porque es despreciable la diferencia entre sets de datos pequeñas.

3 - ¿Por qué en C se puede usar la misma variable tanto para leer de teclado como para enviar por un socket? ¿Esto sería relevante para las aplicaciones c/s?

Ya que utilizamos para esta práctiva punteros a char para el buffer, no tiene más lógica que de estructura contenedora.
Consideramos que no es relevante para las aplicaciones c/s si es utilizado correctamente la memoria asignada.

4 - ¿Podría implementar un servidor de archivos remotos utilizando sockets? Describa brevemente la interfaz y los detalles que considere más importantes del diseño.

Si se podría implementar.
Habría que configurar el cliente para que envíe el nombre del archivo al servidor y el servidor busca el archivo, lo abre y lo manda por red.

5 - Defina qué es un servidor con estado (stateful server) y qué es un servidor sin estado (stateless server)

-- **Statefull server**
El servidor guarda algún estado (información) de un request a otro. De esta manera, el cliente puede mandar menos información por cada request. Este tipo de servidores suelen ser mas simples.

-- **Stateless server**

El servidor no guarda el estado de ningún cliente, cada request es independiente de la previa. Normalmente los clientes de servidores sin estado de deben autenticar por cada request. Aunque requiera la una misma base de información por cada requests, es más robusto a perdidas de conexiones. Ante algún error, no invalida ningún archivo del servidor.