Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA
División de Ciencias Naturales e Ingeniería
Departamento de Matemáticas Aplicadas y Sistemas
$ Licenciatura en Ingeniería en Computación

Sistemas Distribuidos
Práctica 1: sockets y el modelo cliente/servidor
Equipos de 2 personas máximo
Al menos una de las partes (cliente/servidor) en C

Objetivo: Uso de sockets en Linux y conocimiento del modelo cliente servidor.

Actividad durante la sesión

1. **Modificación 1**: **servidor activo**.
   a. Hacer que la versión de sockets TCP, el servidor esté activo siempre. Es decir, agreguen un ciclo para que siempre esté esperando peticiones. La manera de terminarlo será con la combinación de teclas **Ctrl-C**.
2. **Modificación 2: peticiones compuestas**.
   a. En el ejercicio anterior el mensaje que enviaban era una cadena. Sin embargo, ¿cómo enviamos varios datos en un solo mensaje?
   b. Para mandar varios datos debemos "codificarlos" en un solo mensaje del lado del **cliente**, y "decodificarlos" del lado del servidor. **Hay varias maneras de hacerlo**. Por ejemplo, para mandar una solicitud de inscripción necesitamos mandar i) nuestro nombre, ii) una clave de UEA, ii) y un grupo. Para esto haríamos lo siguiente:
      i. Mandar la cadena "`antonio 4604047 CG02C`". Podemos usar la función `sprinf()` para crear esta cadena fácilmente.
      ii. Del lado del servidor debemos implementar una manera de tomar cada dato de la cadena y copiarlo a una variable del tipo adecuado. Una forma muy simple de hacerlo es usando la función `sscanf()`.
   c. Como ejercicio deben implementar un servidor para recibir solicitudes de inscripción.
      i. Cliente: el cliente enviará tres valores codificados en un solo mensaje: i) nombre, ii) clave de UEA y iii) grupo.
      ii. Servidor: el servidor recibirá los tres campos y los agregará a un arreglo para almacenar los tres campos de la solicitud recibida.
         1. Para depurar, después de cada petición el servidor debe mostrar la lista de solicitudes de inscripción.
      iii. Servidor: como respuesta, el servidor regresará dos números, el cupo total del grupo, y el número de la solicitud recibida.
      iv. Cliente: cuando el cliente reciba la respuesta la debe mostrar en la consola.

Actividad para entregar más tarde

Implementación de una mini aplicación UBER

En esta actividad deben implementar una aplicación cliente/servidor que simule la aplicación de UBER para pasajeros solamente. Deben utilizar sockets de tipo TCP para poner utilizar un tunel SSH sí es necesario.

En la aplicación tendrán los siguientes elementos:

- **Solamente tendrán un servidor monohilado** (solamente el *main*), cuya tarea principal será administrar las solicitudes de viajes y autos disponibles. Para llevar un registro de los autos disponibles deben definir un arreglo de valores booleanos de tamaño NUM_AUTOS. Si la posición *i* del arreglo tiene un valor *verdadero*, entonces eso significa que el auto con placas *i* está libre; de lo contrario, está ocupado en un viaje. Por ejemplo, el arreglo:

Libres:	F	V	V	F	V
	0	1	2	3	4

  significa que los autos con "placas" 1, 2 y 4 están libres, mientras que los coches con "placas" 0 y 3 están haciendo un viaje.
- El servidor siempre estará activo. Es decir, la manera de terminarlo será con Ctrl-C. El servidor estará esperando para ofrecer 3 servicios disponibles:
  a. *Estado del servidor*: cuando un cliente solicite el servicio *"estado"*, el servidor debe regresarle **en una sola respuesta** dos valores, el número de viajes realizados y la ganancia total hasta el momento.
     - Recuerden que en *C* solamente pueden enviar una cadena de caracteres, ¿cómo enviar dos o más valores que además no son una cadena?
  b. *Solicitud de viaje*: cuando un cliente solicite el servicio *"viaje"*, el servidor debe elegir uno de los autos disponibles. Si aún quedan autos, le debe regresar al cliente dos valores, las placas del auto y el costo del viaje. Las placas serán un entero entre 0 y NUM_AUTOS-1 (índice del arreglo de autos disponibles), y el costo puede ser un entero aleatorio que ustedes decidan. Si todos los autos están ocupados, el servidor debe regresar simplemente el mensaje *"No hay conductores"*.
  c. *Terminación del viaje*: esta petición solamente la invocan los clientes que pidieron un viaje y consiguieron un auto. Después de que el cliente solicita un viaje y le responden positivamente, deben simular con un *sleep* de unos cuantos segundos que están realizando el viaje. Después de eso, deben solicitar el servicio *"viaje_terminado"* enviando en ese mismo mensaje las placas del auto que usaron. Cuando el servidor reciba ese mensaje, debe marcar en su lista que el auto está disponible nuevamente. En caso de que el cliente no consiga un auto, simplemente debe terminar.
  d. Para propósitos de depuración, después de recibir una conexión el servidor debe imprimir la IP del cliente que hizo la conexión. Deben investigar las dos funciones que se usan para hacer esto.
- **Los clientes**: estos procesos solamente deben hacer una petición y terminar. Para simplificar la implementación, solamente deben implementar un mismo tipo de cliente. Sin embargo, desde la línea de comandos le deben pasar un parámetro para determinar si ese cliente solicitará un viaje (i.e., es la aplicación del pasajero),

o bien, solamente pedirá el estado del servidor (i.e., la aplicación del administrador). Para simular que hacen varias solicitudes, simplemente deben ejecutar el cliente las veces que quieran desde la misma computadora u otras.

Entregables

1. Enviar su código en un archivo ZIP.
2. Deben entregar un reporte (**no** poner el documento dentro del ZIP) que contenga la explicación de las partes más importantes de su código (e.g., cómo resolvieron enviar dos valores en cada mensaje). También deben incluir capturas de pantalla de un ejemplo donde los procesos servidor y cliente(s) estén en computadoras diferentes.
3. Deben enviar un video corto donde se muestre al servidor recibiendo peticiones desde otras computadoras en red. No es necesario poner audio en el video.