Práctica 3. XDR

Arquitectura y Tecnología de Computadores. Sistemas Distribuídos

1. Ejercicios a realizar

Los ejercicios a realizar consisten básicamente en los propuestos durante la sesión 3 de prácticas, con ligeras variaciones. Se detallan a continuación.

1.1. Cliente/servidor para transmisión de datos XDR simples

Partiendo del fichero `tipos.x` suministrado en la sesión 3, escribir un cliente y un servidor que intercambien datos de los tipos declarados en dicho fichero.

El cliente recibirá por línea de comandos el nombre de la máquina y el número de puerto en los que escucha el servidor. Tras conectar con éxito con ese socket, realizará al usuario una serie de preguntas para inicializar una variable de cada uno de los tipos declarados en `tipos.x`, en el orden en que aparecen allí declarados. En el array de longitud variable el usuario podrá elegir el número de datos. En la unión podrá elegir el discriminante y el valor asociado. Observar que *todas las variables* deben ser rellenadas con valores suministrados por el usuario, y no de forma estática en el código, como era el caso de la sesión de prácticas. Una vez inicializadas todas las variables, haciendo uso de filtros XDR se enviarán al servidor (en el orden de declaración en `tipos.x`) y se desconectará.

El servidor recibirá por línea de comandos el número de puerto en que debe escuchar. Creará un socket de escucha y se bloqueará en un `accept()` a la espera de clientes. Cuando llegue uno, usará filtros XDR para recibir y decodificar los datos que éste le envíe, y seguidamente mostrará por pantalla los valores recibidos. Tras esto, cerrará el socket de datos y volverá a la espera de otro cliente.

2. Cliente/servidor para la transmisión de datos complejos

Escribir un fichero con el tipo XDR adecuado para implementar una lista enlazada[1], en la que cada nodo contenga un entero.

Escribir un cliente similar al del apartado anterior, que conecte con un servidor dado por línea de comandos y pida al cliente una lista de números. Esto se realizará mediante un bucle en el que, partiendo de una lista vacía, para cada dato suministrado por el usuario se crea un nuevo nodo y se añade a la lista (es indiferente si se añade al principio o al final), hasta que el usuario introduzca cero como dato. Este caso no se introduce en la lista, sino que da por terminado el bucle. El cliente enviará esta lista al servidor haciendo uso del filtro XDR y finalizará su ejecución.

El servidor creará un socket de escucha y quedará a la espera de clientes en una llamada a `accept()`. Cuando reciba uno, hará uso del filtro XDR para recibir la lista completa, y seguidamente imprimirá los datos almacenados en cada nodo (mediante un bucle que finaliza cuando detecte un puntero nulo en la lista enlazada). Seguidamente liberará la memoria ocupada por la lista (haciendo uso de `xdr_free()`), cerrará el socket de datos y volverá a la espera de nuevos clientes en el socket de escucha.

[1] No es válido programarlo en forma de array de longitud variable

2.1. Caso especial

Observar que hay un caso particular que puede plantear dudas, y es aquél en el que el usuario introduce un cero como primer dato. De acuerdo con lo enunciado anteriormente, este primer cero no debería formar parte de los datos, sino indicar que no van a introducirse más; por tanto la lista generada debería estar vacía. No obstante, como la implementación de este caso es algo compleja, se deja a la voluntad del alumno el que elija una de las dos posibilidades siguientes, para el caso de que el usuario introduzca un cero como primer dato:

1. **Versión sencilla:** Este caso se trata de forma especial. Cuando el cero se introduce como primer dato, no se considera indicador de final, sino un dato más. Si el usuario mete dos ceros seguidos, se enviaría una lista con un solo nodo, que contendría el primer cero.
2. **Versión compleja:** Este caso no se trata de forma especial. Un cero como primer dato implica que la lista no tiene datos. Cliente y servidor deben manejar este caso adecuadamente.

Se valorará positivamente si se elige implementar la modalidad más complicada.

3. Aspectos formales

Debes crear una carpeta llamada `p3` y dentro de ella dos carpetas llamadas `e1` y `e2` , una para cada uno de los ejercicios anteriores. Cada carpeta debe contener el código fuente (con los nombres de ficheros que detallaremos seguidamente) y alguna forma automatizada de compilación, ya sea en forma de *script* o de `Makefile`. El código fuente debe ser C estándar y debe compilar y funcionar al menos en las dos máquinas de prácticas `sirio` y `orion`.

Los nombres de los ficheros fuente serán los siguientes:

Ejercicio 1 `tipos.x`, `cliente.c` y `servidor.c`

Ejercicio 2 `tipos_lista.x`, `cliente-lista.c` y `servidor-lista.c`

3.1. Entrega

Cuando el código haya sido probado, se borrarán los ejecutables, dejando sólo el código fuente y el *script* o `Makefile` de compilación, y se empaquetarán todas las carpetas en un único archivo `tar` comprimido. Para esto, debes situarte en la carpeta "padre" de la carpeta `p3` y desde ella teclear:

```
$ tar czvf p3.tar.gz p3
```

El resultado será el fichero `p3.tar.gz`, que seguidamente deberás firmar digitalmente con el comando[2]:

```
$ gpg --sign p3.tar.gz
```

El resultado será un archivo `p3.tar.gz.pgp` que entregarás a través del formulario Web en la página de la asignatura. Si lo deseas, además de firmarlo puedes cifrarlo para tu profesor de prácticas, mediante el comando:

```
$ gpg --sign --encrypt -r jldiaz@uniovi.es -r ariasjr@uniovi.es p3.tar.gz
```

[2]Deberás efectuar esta operación en la máquina en la que hayas creado tu pareja de claves GPG