Laboratorio N.º 3

SISTEMAS Distribuidos

Primer cuatrimestre de 2020

Departamento de Ciencias e Ingeniería de la Computación - Universidad Nacional del Sur

Alumnos

- **Garat Manuel** **Registro: 110996**
- **Gianni Lucas Cerdá** **Registro: 103138**

1. ¿Para qué se utilizan los archivos *services, hosts, networks y protocols*? ¿En cuáles sistemas operativos se utilizan?

Archivo Services:

El archivo *services* contiene información sobre los nombres de los servicios registrados en la máquina (Local Services), número del puerto asociados a cada servicio, con que tipo de protocolo trabaja ese servicio (UDP/TCP) y cualquier alias asociado. Se encuentran tanto en sistemas operativos Windows como en UNIX y SUN en los siguientes directorios.

- Windows: *C:\Windows\System32\drivers\etc\services*
- UNIX - SUN : */etc/services*

Archivo Hosts:

El archivo *hosts* es usado por el sistema operativo para guardar la correspondencia entre dominios de Internet y las direcciones IP a las que se asocian. Antiguamente el archivo host era el único encargado de hacerlo, pero dejó de utilizarse cuando Internet empezó a crecer, pasando a usar servidores de resolución de nombres (DNS).En muchos sistemas operativos este método es usado preferentemente respecto al DNS.

Se encuentran tanto en sistemas operativos Windows como en UNIX y SUN en los directorios:

- Windows: *C:\Windows\System32\drivers\etc\hosts*
- UNIX - SUN : */etc/hosts*

Archivo Networks:

El archivo networks permite definir un alias a las redes, de manera similar al archivo hosts en las máquinas distribuidas a lo largo de la red. El archivo se conforma de líneas de texto donde se guarda un nombre de red , su dirección, y los alias que le fueron asignados. Estos alias se convierten en una guía para la administración de la red, se mantiene la estructura independientemente de que se cambie algún componente de hardware como podría ser un router.

Se encuentran tanto en sistemas operativos Windows como en UNIX y SUN en los directorios:

- Windows: *C:\Windows\System32\drivers\etc\networks*
- UNIX -SUN : */etc/networks*

Archivo Protocols:

El archivo *protocols* describe los distintos protocolos de transporte que están disponibles en el sistema de red. En en archivo se incluye el nombre nativo del protocolo, su número oficial tal como iría en la cabecera IP y alias en caso de tenerlos. Los nuevos protocolos que van surgiendo son agregados a este archivo siempre y cuando sean asignados oficialmente por la IANA.

Se encuentran tanto en sistemas operativos Windows como en UNIX y SUN en los directorios:

- Windows: *C:\Windows\System32\drivers\etc\protocols*
- UNIX Y SUN : */etc/protocols*

2. Enumere 3 puertos asignados a servicios específicos.

Algunos puertos bien conocidos (well-known port) son :

1. **80 HTTP** HyperText Transfer Protocol (Protocolo de Transferencia de HiperTexto)
2. **115 SFTP** Protocolo de transferencia de archivos seguros .
3. **22 SSH** (SSH,sftp,scp)

3. ¿Para qué sirve el netstat*?. ¿ Brinde dos ejemplos en los cuales se brinde diferente tipo de información?*

El comando *netstat* lista todas las conexiones activas de la computadora. Esto incluye las entrantes y las salientes.

La información que resulta del uso del comando incluye el protocolo en uso, las tablas de ruteo, las estadísticas de las interfaces y el estado de la conexión.

Ejemplos:

- Desplegar los miembros del grupo multicast → *~$ netstat --groups*
- Mostrar las tablas de ruteo del kernel → *~$ netstat --route*

4. ¿Cómo conoce que el portmapper (*rpcbind*) está activado?

Para conocer si el portmapper está activo, entonces ejecuta el siguiente comando:

- *~$ systemctl status rpcbind*
- *~$ service portmap status* (Para sistemas operativos más antiguos)

5. ¿Es posible pasar múltiples argumentos con SunRPC?

No es posible pasar múltiples argumentos con SunRPC. Como alternativa a este problema, se almacena los argumentos en una estructura (equivalente al struct de C) , que luego se utilizan desde el servidor. El único conveniente que tiene esta metodología es que se genera un archivo XDR para determinar cómo acceder a los datos que se encuentran en dicha estructura. Esto agrega overhead a la compilación y al intercambio de mensajes.

6. ¿Para qué se utiliza el comando *make*? ¿Qué relación existe con el tiempo?

Make es una herramienta de gestión de dependencias de los archivos que forman parte del código fuente de un programa. Su principal uso es el de controlar la compilación y la recompilación de forma automática. También, puede utilizarse en escenarios donde se requiera actualizar automáticamente un conjunto de archivos a partir de otro cada vez que éste cambie.

Este comando funciona recursivamente, partiendo del objetivo final. Para cada posible fichero a regenerar (fichero objetivo), primero ser reconstruyen , si es necesario, los ficheros de los que depende de manera inmediata y directa, y a continuación se comprueba la fecha y hora de actualización (que pueden haber cambiado si se ha regenerado alguno de ellos) para ver si el fichero objetivo está al día.

La relación que existe con el tiempo es muy importante ya que se necesita saber su fecha de actualización cuando se necesita recompilar. Antes de recompilar, se verifica la fecha y hora del archivo a actualizar y si esta es posterior a la fecha y hora del conjunto de archivos que genera, los recompila, en caso contrario, deja la versión existente.

Make realiza una serie de pasos para determinar si debe compilar de nuevo o no un archivo:

- Detecta los ficheros no actualizados mediante la comparación de timestamps ó estampillas de tiempo.
- En caso que necesite recompilar, debe tener información acerca de las acciones para regenerar los archivos. Esta información se suministra en

un fichero denominado comúnmente “makefile”, usando una notación especifica basada en reglas.

- Cada regla makefile contiene dependencias de un archivo respecto a los demás, y los comandos para crearlos.

7. ¿Qué es más conveniente para soportar tolerancias a fallas un servidor sin estado o con estado? Justifique su respuesta.

- La mayor desventaja que poseen los servidores con estado es que si se produce una caída del sistema o crashea, el servidor debe recuperar toda la información de tal forma como estaba antes de caerse (rollback), por lo tanto, necesita diferentes tipos de herramientas complejas para su recuperación.
- En un diseño sin estado no requiere de este tipo de medidas especiales después de caerse, simplemente arranca de nuevo esperando peticiones de los clientes como si el servidor hubiera arrancado desde cero. Por lo tanto, el servidor sin estado puede o tiene mayores tolerancias a fallas que un servidor con estado.

8. Problema: Un cliente/par desea consultar a otra máquina las siguientes solicitudes:

El sistema operativo instalado.

Fecha y hora del sistema con el siguiente formato Mon Oct 26 18:10:27

a) Debe implementarse utilizando RPC como mecanismo de comunicación.

- La implementación se encuentra en la carpeta: **entrega/fuentes/9/a**

b) Debe implementarse utilizando sockets con conexión como mecanismo de comunicación.

- La implementación se encuentra en la carpeta: ***entrega/fuentes/9/b***

c) Debe implementarse utilizando sockets sin conexión como mecanismo de comunicación.

- La implementación se encuentra en la carpeta: ***entrega/fuentes/9/c***

d) Si usted hubiera tenido que elegir la forma de implementarlo, ¿Cuál mecanismo de comunicación hubiera elegido? ¿Porqué?

En caso de haberlo implementado, hubiese sido preferible usar RPC, ya que uno podría abstraerse de cuestiones de implementación específicas de cada arquitectura (gracias a que todo mensaje se encapsularía con la ayuda de los stub cliente y servidor). También se tendrían mayores facilidades de conectividad ya que no habría que preocuparse por abrir puertos entre clientes y servidores como podría ocurrir en el empleo de sockets.
Además, personalmente, nos resulta más modular e intuitivo la implementación de RPC.