UNIVERISDAD DE COSTA RICA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA

CI-0123 Proyecto integrador de redes de comunicación de datos y sistemas operativos

Prof. Francisco Arroyo

Prof. Tracy Hernández

Grupo 01

Diseño del protocolo de red

Trabajo grupal de toda la clase

Ciclo 1 - 2025

Conexión y desconexión

Mecanismo de detección por UDP

El mecanismo de detección UDP tiene como actores el servidor y el tenedor, la intención es no tener la necesidad de especificar en el código o en tiempo de ejecución dónde y en qué puerto se encuentra escuchando un servidor o tenedor.

Esto se logrará a través de un aviso que realizaría el servidor y el tenedor cada vez que se encienden, ya que haría un aviso por UDP a todos los que se encuentren en la red con los siguientes comandos:

Desde un servidor:
“BEGIN /ON/{tipo de máquina (SERVIDOR)} {ip de máquina} {puerto} END”
Ej: BEGIN/ON/SERVIDOR/192.168.1.10/8080/END

Desde un tenedor:
“BEGIN /ON/{tipo de máquina (TENEDOR)}/END”
Ej: BEGIN/ON/TENEDOR/END

Esto con el fin de avisar que se encuentra disponible y atento a una conexión por TCP, lo haría una sola vez al levantarse y posteriormente se pondría a escuchar por peticiones TCP, para no tener espera activa. Luego de recibir el mensaje UDP, tenemos dos posibles escenarios:

1. Tenedores envían y no hay servidores: En este caso el tenedor envía primero el mensaje UDP y ningún servidor lo recibe, pero cuando se levante el servidor va a avisar y el tenedor lo va a detectar y solicitar su conexión.
2. Servidores envían y no hay tenedores: En este caso el servidor envía primero el mensaje UDP y ningún tenedor lo recibe, pero cuando se levante el tenedor va a avisar y el servidor lo va a detectar y va a reenviar

su mensaje UDP con su información, para que el tenedor que se acaba de levantar lo detecte.

Conexión por TCP

Luego de recibir el mensaje UDP, los tenedores guardan la información del servidor (IP y puerto) como una opción válida para establecer conexiones TCP, además realizan la primera conexión con el fin de solicitar los objetos disponibles con el siguiente comando al servidor: “BEGIN/OBJECTS/END”.

El resultado de esta petición se utilizará para almacenar en una lista de servidores sus objetos disponibles esto con el fin de validar o rechazar futuras solicitudes HTTP de los clientes.

Reconexión y manejo de desconexión

Si un servidor se apaga o desea desconectarse voluntariamente, debe enviar un nuevo mensaje UDP de cierre, indicando su desconexión:

“BEGIN/OFF/SERVIDOR/{ip}/{puerto}/END”.
Ej: BEGIN/OFF/SERVIDOR/192.168.1.10/8080/END

Los tenedores que reciban este mensaje deben eliminar o marcar como inactivo ese servidor en sus listas de servidores disponibles.

Si el servidor se apaga abruptamente (sin poder enviar el mensaje UDP de cierre), los tenedores deben aplicar una lógica de reconexión o detección de fallos por timeout (pueden ser 5 segundos) y una cantidad de intentos (podrían ser 3), por lo que si no recibe respuesta luego de varios intentos de conexión TCP, eliminan al servidor y su lista de objetos de forma automática.

Administración de recursos (memoria y almacenamiento)

El protocolo implementa un uso eficiente de memoria y almacenamiento para garantizar el rendimiento del sistema y la disponibilidad de los recursos. Las decisiones de diseño en esta área se enfocan en el uso temporal y persistente de datos, así como en evitar el consumo excesivo de recursos en cada componente.

Memoria

- **Gestión de listas dinámicas:** Tanto el servidor como el tenedor mantienen listas de figuras en estructuras dinámicas en memoria (como “vector” o “queue”), permitiendo su crecimiento o eliminación de manera eficiente conforme se actualizan los recursos disponibles.
- **Desasignación automática:** Se emplean mecanismos de liberación de memoria como destructores o limpieza manual luego de desconexiones, de modo que no se acumulen referencias a servidores caídos o figuras inexistentes.
- **Timeout y retry limitados:** Para evitar esperas activas prolongadas que consumen CPU y memoria, los intentos de reconexión están limitados (por ejemplo, a 3 reintentos). Después de esto, los datos relacionados se eliminan de la memoria.

Almacenamiento

- **Carga bajo demanda:** Las figuras solicitadas se leen desde disco solo cuando el cliente las requiera, evitando mantener archivos innecesariamente cargados en memoria.
- **Evicción de archivos temporales:** Si se utilizan archivos temporales para procesar solicitudes (por ejemplo, al construir una respuesta a partir de múltiples fuentes), estos son eliminados inmediatamente después de ser enviados al cliente.
- **Persistencia de catálogos:** Las listas de figuras disponibles por servidor pueden almacenarse de forma local en archivos simples (archivos .txt, por

ejemplo) para mantener respaldo entre sesiones, aunque siempre se validan con la lista real del servidor al momento de una nueva conexión.

Instrucciones

En esta sección se definen los comandos e instrucciones utilizadas en la comunicación entre los distintos actores del sistema: servidores, tenedores y clientes. Estas instrucciones permiten la consulta de recursos, la validación de solicitudes y la transferencia de datos dentro del protocolo de red colaborativo.

Anuncio de disponibilidad (UDP)

Formato servidor: BEGIN/ON/SERVIDOR/{ip}/{puerto}/END

Ejemplo: BEGIN/ON/SERVIDOR/127.0.0.1/1234/END

Formato tenedor: BEGIN/ON/TENEDOR/END

Este mensaje permite a los actores anunciar su presencia en la red al iniciarse.

Desconexión (UDP)

Formato: BEGIN/OFF/SERVIDOR/{ip}/{puerto}/END

Ejemplo: BEGIN/OFF/SERVIDOR/127.0.0.1/1234/END

El servidor notifica su salida voluntaria del sistema.

Solicitud de lista de objetos (TCP)

Formato: BEGIN/OBJECTS/END

El tenedor solicita al servidor la lista de objetos (por ejemplo, dibujos disponibles). El servidor responde con una lista textual, separada por saltos de línea.

Formato: BEGIN/OK/OBJECT\nOBJECT1/END

Ejemplo: BEGIN/OK/ballenita\nballenota\nBallena/END

En caso contrario, responde con un mensaje de error.

BEGIN/ERROR/104/El servidor de dibujos se encuentra vacío/END

Solicitud de recurso por parte del cliente

Formato: BEGIN/GET/{nombre_de_dibujo}/END

Ejemplo: BEGIN/GET/circle/END

El tenedor valida si el recurso existe en algún servidor válido. Si lo encuentra, lo solicita al servidor y lo devuelve al cliente. Si no, responde con un mensaje de error.

Respuesta de recurso

Si el recurso existe, se devuelve el contenido del archivo .txt. Para la implementación se recomienda que se tome como el contenido del dibujo, todo lo que esté entre OK/ y /END.

Formato: BEGIN/OK/contenido del dibujo/END

Ejemplo: BEGIN/OK//END

Si no existe:

BEGIN/ERROR/201/El dibujo solicitado no fue encontrado/END

Errores y formato de mensajes de error

Formato general: BEGIN/ERROR/{Código o Tipo}/{Mensaje}/END

Ejemplo: BEGIN/ERROR/100/No se puede establecer conexión con el servidor/END

Ejemplos:

- ERROR 100: No se puede establecer conexión con el servidor.

Observaciones

Todas las instrucciones deben respetar estrictamente el formato del protocolo. Además, los mensajes mal formateados deben ser descartados y notificados a los desarrolladores como error de tipo PROT, y la interacción entre cliente-tenedor y tenedor-servidor debe ser robusta ante fallos, permitiendo reconexión o la exclusión de nodos inactivos sin afectar la disponibilidad general del sistema.

Manejo de errores

En la presente sección se especifica cómo actuar ante errores que pueden surgir al trabajar con comunicación entre un cliente, un tenedor y un servidor. Cabe destacar que hay errores de servidor, que funcionan tanto para fork como para server, en los que esto es aplicable, es indicado en la sección de caso para cada error.

Tipos de errores

1. **Servidor no disponible o alcanzable**
    - **Caso:** No se logra establecer conexión, ya sea porque el puerto que se utiliza es inválido o la IP; aplicable para Fork y Sever.
    - **Respuesta:** Enviar un mensaje de error al usuario y no es necesario detener el funcionamiento, simplemente se debe descartar el servidor no disponible de su tabla de servidores y esperar por futuras conexiones, si el cliente pide una lista de objetos se entrega vacía. Si hay más de un servidor, se deben aceptar solicitudes únicamente al servidor o servidores cuya conexión está funcionando.
2. **Pérdida abrupta de conexión al servidor**
    - **Caso:** Se pierde conexión con un servidor sin haber comunicado esto con el mensaje BEGIN/OFF/SERVIDOR/{ip}/{puerto}/END o con un Fork, como se estipula en el protocolo, y tras hacer más intentos de reconexión no se logra una conexión, significa que sucedió un error; aplicable para Fork y Sever.

- **Respuesta:** Si al hacer más intentos (por el momento 3) de reconexión, todos terminan en un timeout, significa que hubo una desconexión abrupta del servidor. Enviar un mensaje de error al usuario y no es necesario detener el funcionamiento, simplemente se debe descartar el servidor no disponible de su tabla de servidores y esperar por futuras conexiones, si el cliente pide una lista de objetos, se entrega vacía. Si hay más de un servidor, se deben aceptar solicitudes únicamente al servidor o servidores cuya conexión está funcionando.

3. **Servidor de dibujos vacío**
    - **Caso:** Al hacer la solicitud BEGIN/OBJECTS/END, el servidor de dibujos no devuelve una lista de dibujos, significa que este está vacío.
    - **Respuesta:** No se detiene el programa, sin embargo, se debe de comunicar al usuario que el servidor de dibujos está vacío por lo tanto ningún dibujo que solicite se podrá encontrar.
4. **Timeout**
    - **Caso:** Se envía una solicitud al servidor o Fork y tras una cantidad estipulada de tiempo de 5 segundos no se recibe respuesta, se declara un timeout de espera; aplicable para Fork y Sever.
    - **Respuesta:** Si al envíar una solicitud al Fork o Server, se da un timeout de una cantidad de 5 segundos, significa que algo salió mal. Esto se maneja tratando de reconectar 3 veces, si las 3 veces se obtiene timeout, el servidor se declara como que tuvo pérdida abrupta de conexión (error 2).
5. **Solicitud inválida de cliente**
    - **Caso:** Es un error de formato, el cliente hace una solicitud que no es coherente con el tipo de datos ({figura}.txt).
    - **Respuesta:** No es necesario detener el funcionamiento del programa, sin embargo, es necesario comunicar al cliente el formato de solicitud de dibujo y esperar otra solicitud.
6. **Dibujo no encontrado**

- **Caso:** Un cliente solicita un dibujo de la forma {figura}.txt y no se encuentra dentro del servidor en el que lo solicitó.
- **Respuesta:** No es necesario detener el funcionamiento del programa, sin embargo, es necesario comunicar al cliente que no se encontró el dibujo y esperar otra solicitud.

7. **Pérdida abrupta de conexión al cliente**
    - **Caso:** Se pierde la conexión al cliente de manera abrupta, ya sea por desconexión o un cierre abrupto.
    - **Respuesta:** No es necesario detener el funcionamiento del programa, sin embargo, si el cliente hizo una solicitud y para el momento en el que esta fue respondida, la conexión ya había fallado, se debe de descartar o ignorar la respuesta a dicha solicitud.
8. **Error en mensaje de acuerdo al protocolo**
    - **Caso:** Se da si de acuerdo al protocolo, hay un mensaje que no está bien formateado. Se puede dividir en cada formato para cada tipo de mensaje. Por ejemplo que el formato del mensaje sea: “BEGIN /ON {tipo de máquina (SERVIDOR, TENEDOR)} {ip de máquina} {puerto} END”, y el mensaje sea “BEGIN /ON SERVIDOR 192.168.1.10 END”, falta el puerto, este tipo de error cubre esto.
    - **Respuesta:** Al ser un error interno, se debe de comunicar a los desarrolladores del programa para arreglarlo, sin el correcto formato de mensajes, el programa no puede funcionar correctamente.

Códigos de error

Se propone asignar un código o número para identificar cada error dentro del programa, para propósitos de debug y orden para la clasificación de estos. Para clasificarlos, se tienen errores de servidor/fork, error de cliente y error de protocolo.

Formato de código de error	Categoría
1xx	Servidor/Fork
2xx	Cliente
3xx	Protocolo

Tabla 1: Clasificación de códigos de error

En la columna de formato de código de error las “x” representan los otros números que conforman los códigos de error. Por ejemplo: Error 100, error 301, … A continuación se definen los errores, con su código y mensaje, algunos errores mencionados en la sección anterior se dividen en errores de servidor y errores de tenedor, sin embargo, son los mismos que los definidos anteriormente. La columna de *Número de error* se refiere al número de error definido en la sección de tipos de errores, la columna de *Error* es una pequeña descripción del error, la columna de *Código* es el código numérico asociado al error y la columna de *Mensaje* es un mensaje con el que se le puede comunicar al usuario y desarrollador del error.

N° error	Error	Código	Mensaje
1	Servidor no alcanzable	100	No se puede establecer conexión con el servidor.
1	Fork no alcanzable	101	No se puede establecer conexión con el tenedor.
2	Pérdida abrupta de conexión al servidor	102	Sucedió una pérdida abrupta de conexión al servidor.
2	Pérdida abrupta de conexión al tenedor	103	Sucedió una pérdida abrupta de conexión al tenedor.
3	Servidor de dibujos vacío	104	El servidor de dibujos se encuentra vacío.
4	Timeout de servidor	105	Ha sucedido un timeout de solicitud al servidor.
4	Timeout de tenedor	106	Ha sucedido un timeout de solicitud al tenedor.
5	Solicitud inválida de cliente	200	La solicitud del cliente tiene un formato inválido.
6	Dibujo no encontrado	201	El dibujo solicitado no fue encontrado.
7	Pérdida abrupta de conexión al tenedor	202	Ha sucedido una pérdida abrupta de conexión al cliente.
8	Error en mensaje por protocolo	300	Ha sucedido un error en el formato del mensaje: {mensaje}.

Tabla 2: Errores

Estructura de mensajes de error

Para comunicar que ha sucedido un error se puede utilizar la información de la Tabla 2, la estructura de mensaje es la siguiente: BEGIN/ERROR/{Código de error}/ {Mensaje}/END. Por ejemplo, si se tiene un error de servidor no alcanzable, entonces el mensaje de error sería el siguiente: ERROR 100: No se puede establecer conexión con el servidor.