Laboratorio N.º 2

SISTEMAS Distribuidos

Primer cuatrimestre de 2020

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

Alumnos

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

1. Si el paradigma de comunicación es asincrónico, ¿es también persistente? Explique su respuesta y brinde un ejemplo.

Comunicación Asincrónica:
Es aquella en la que el remitente continúa inmediatamente después de que ha pasado su mensaje para la transmisión. Es decir, el mensaje debe ser almacenado de manera temporal inmediatamente bajo la supervisión del middleware.

Comunicación persistente:
El middleware de comunicación almacena el mensaje que ha sido presentado para transmitirse el tiempo que tome entregarlo al destinatario. En este caso el middleware almacenará el mensaje en uno o varios de los lugares del almacenamiento. Como consecuencia, no resulta necesario que la aplicación remitente continúe con la ejecución después de presentar el mensaje. De igual manera la aplicación destinataria no necesita estar en ejecución cuando el mensaje se presenta.

En la práctica existen diferentes combinaciones según el tipo de comunicación que se necesite para la ocasión, la más popular es la combinación de una comunicación persistente y asincrónica. Esta combinación se presenta ya que en el modelo transitorio existe la posibilidad de que los datos se pierdan debido al asincronismo con el que trabajan los procesos da lugar que no siempre se estén ejecutando a la hora de intercambiar un mensaje t y por lo tanto el mismo se pierda.

Con un modelo de comunicación persistente, uno de los dos procesos o ambos pueden no estar en ejecución y en tal caso se asegura que el mensaje se almacenará hasta que puede ser entregado.

Un ejemplo es un sistema de emails, en donde el remitente/emisor envía un mensaje y continúa realizando alguna otra tarea y no espera a que el mail se entregue, por lo tanto, este ejemplo es una comunicación persistente, ya que si el emisor no está activo luego de enviar el mail y el receptor/destinatario no está en ejecución, esté se perdería, otro ejemplo análogo podría ser una transferencia bancaria.
Por último y para resumir, en un esquema asincrónico, la comunicación suele ser persistente para asegurar la entrega del mensaje, aunque dependiendo de la práctica esta condición no es necesaria.

2.Dos divisiones de la armada de Rocolandia, Rojos y Azules, están acampados en 2 colinas cercanas. En el valle que está entre las 2 colinas está ubicada la división enemiga Zapatilandia. Las divisiones de Rocolandia están seguras donde están ubicados sus campamentos, y pueden intercambiar confiadamente mensajes a través del valle. ¿Pueden llegar a un acuerdo para atacar al enemigo en forma conjunta si el modelo de sistema es asincrónico? Justifique.

Sí, se puede utilizando un protocolo de comunicación de tres vías ya que el pasaje de mensajes a través del valle es confiable, pero se deben considerar ciertas condiciones como el delay o latencias propias de la comunicación remota.
Sin embargo, para que tenga éxito el ataque en conjunto, es necesario que acuerden las siguientes reglas:

i). La división Roja le envía un mensaje a la división Azul con la hora del ataque, esto se debe realizar con un tiempo de anticipación considerable, sabiendo de antemano que el envío del mensaje no es instantáneo, considerando las latencias que pueden surgir.

ii). La división Azul, tras recibir el mensaje con la hora estipulada, le confirma la hora de ataque.

iii). La división Roja, le ratifica que recibió el mensaje por parte de la división Azul.

Posibles problemas que pueden ocurrir:

El mensaje con la hora de ataque por parte de la división Roja llega al receptor, i.e., la división Azul pero un tiempo posterior a la hora del ataque previsto.

Otro inconveniente, podría ser un ataque sorpresa por parte del enemigo a alguna de las dos divisiones luego de la confirmación del mensaje, en consecuencia, unas de las divisiones pelearía sin apoyo.

Por último, aunque todas las divisiones hayan acordado la misma hora con éxito, diferencias en los relojes de cada una van a provocar que el ataque no sea perfectamente sincronizado.

3. Considere un cliente de correo electrónico y conteste las siguientes preguntas. Explique las respuestas.

a) ¿Cuál modelo de comunicación le parece el más apropiado?

b) ¿Utilizaría algún nivel de sincronismo?

a) Para un cliente de correo electrónico el modelo más apropiado es el asincrónico y persistente. (Como se describió en el ejemplo del Ejercicio 1)

Asincrónico ya que el emisor no necesita esperar a que el receptor reciba el mail. También, persistente porque el mail no se perderá si es que el emisor o el receptor no se encuentran en ejecución en la comunicación.

b) Se podría sincronizar con etiquetas de tiempo el orden de los mensajes en caso de que lleguen en forma desordenada.

4. Realizar un experimento para obtener el tiempo de ejecución en la escritura de un pipe y en la escritura de un archivo. El nombre del archivo fuente debe comenzar con la palabra *escribir*. Explique brevemente los resultados obtenidos. La precisión de las mediciones tienen que estar en **microsegundos**.

La escritura en el pipe se realiza más rápidamente que la escritura en el archivo ya que la de la primera se realiza en memoria principal mientras que la segunda se realiza en un almacenamiento secundario.

```
> ./escribir
 El tiempo para escribir en un archivo es: 64 microsegundos
 El tiempo para escribir en un pipe es: 14 microsegundos
> ./escribir
 El tiempo para escribir en un archivo es: 15 microsegundos
 El tiempo para escribir en un pipe es: 8 microsegundos
```

5. Realizar un experimento para obtener el tiempo de ejecución en la creación de una cola de mensajes y la escritura de un mensaje. El nombre del archivo fuente debe *colamen*. Explique brevemente los resultados obtenidos. La precisión de las mediciones tienen que estar en **nanosegundos**.

La escritura en la cola de mensajes se realiza más rápidamente que la creación de la misma ya que para crearla se debe acceder al disco para obtener la clave de la misma, mientras que la escritura se realiza en memoria.

```
El tiempo de creación de la cola de mensajes es: 42464 nanosegundos
El tiempo de escritura en la cola de mensajes es: 10651 nanosegundos
> ./colamen
El tiempo de creación de la cola de mensajes es: 22096 nanosegundos
El tiempo de escritura en la cola de mensajes es: 12372 nanosegundos
```

6. ¿Qué es el binder? Enumere las formas de ubicar un binder. Explique la sobrecarga que se produce en la comunicación para cada uno de los casos.

El servicio de resolución de nombres se denomina Servidor de Nombres (en inglés también se lo conoce como binder, ya que a la traducción de nombres se le denomina binding). Cuando un cliente necesita conocer la dirección de cualquier servidor, debe preguntarle al servidor de nombres. El binder mantiene una tabla llamada "Binding Table" que es un mapeo de los nombres de interfaces de los servers a sus locaciones. Todos los servers deben registrarse ellos mismos con el binder como parte de su proceso de inicialización. Desde luego, el servidor de nombres residirá en alguna máquina de dirección bien conocida.

Las formas de ubicar un binder son las siguientes:

- **Ubicación conocida:**

La dirección de binder es fija y conocida previamente por los nodos. Se envían las peticiones directamente a éste. En este caso, no existe sobrecarga generada debido a la ubicación de binder ya que éste se encuentra fijo en una dirección conocida por los clientes y servidores de forma anticipada. En caso de reubicación del binder, este notifica su nueva ubicación a cada uno de los nodos, generando sobrecarga.

- **Ubicación proporciona por sistemas operativos:**

Que los sistemas operativos de los clientes y de los servidores sean responsables de proporcionar dinámicamente a sus procesos la dirección del binder. Esto puede hacerse mediante variables de entorno. Si se reubica el binder, debe actualizarse la dirección en la variable de entorno. Si se reubica el binder, debe actualizarse la dirección en la variable de entorno del sistema operativo. Respecto a la sobrecarga debido a la ubicación del binder, en este caso se da debido a que el SO debe comunicarse con los procesos para indicar la dirección del binder.

- **Ubicación desconocida**:

Cuando el cliente o un servidor se enciende, difunde un mensaje para toda la red para localizar al binder. Cuando un proceso binder reciba este mensaje, contesta al emisor con su dirección. Mediante este sistema, el binder puede ejecutarse siempre sobre cualquier computadora y reubicarse dinámica y fácilmente. Debido a esta facilidad, la sobrecarga es mucho mayor que en los casos anteriores pues el envío de mensajes para localizar el binder genera una mayor cantidad de tráfico por la red.
El servidor de nombres puede estar ubicado en un servidor que no sea destinado para esta tarea específicamente, o bien estar localizado en una máquina que actúe como servidor dedicado para la resolución de nombres.

Debido a la importancia de la utilidad del binder se recomienda que se encuentre replicado para evitar que se convierta en cuello de botella y por lo tanto tenga la capacidad de tolerar fallas.

7. Considere la información de la figura 1

a) ¿Qué representa el código de la figura 1? Explique cada uno de los elementos.

```
#define TAMANO_BLOQUE 512
struct stat_archivo
{
int hubo_error;
char descripcion_error[255];
int       st_mode;        /* protección */
int       st_nlink;       /* nro de enlaces físicos */
int       st_uid;       /* ID del usuario propietario */
int       st_gid;       /* ID del grupo propietario */
long int      st_size;      /* tamaño total en bytes */
unsigned long st_blksize;    /* tamaño de bloque
para el sistema. de archivo E/S */
unsigned long st_blocks;    /* nro de bloques asig */
};
struct archivo_escritura
{
      char nombre[512];
      long desplazamiento;
      int datos[TAMANO_BLOQUE];
      int datos_size;
      int hubo_error;
      char descripcion_error[255];
};
struct archivo_lectura
{
      char nombre[512];
      long desplazamiento;
      int datos_size;
};
struct archivo_atributos
{
      char nombre[512];
      int modo;
};

program FILESERVER
{
      version PRIMERA
      {
          error escribir(archivo_escritura archivo) =1;
          archivo_escritura leer(archivo_lectura entrada) =2;
          stat_archivo obtener_atributos(camino_t camino) =3;
          error modificar_atributos(archivo_atributos archivo) =4;
          error borrar_archivo(camino_t camino) =5;
          error crear_archivo(camino_t camino) =6;
          cadena directorio_actual() =7;
          error cambiar_directorio(camino_t nombre) =8;
          error crear_directorio(archivo_atributos directorio) =9;
          entradas_directorio listar_directorio(camino_t nombre) =10;
          error renombrar(cambiar_nombre nombres) =11;
      } = 1;
```

```
struct camino_t
{
  char nombre[512];
};
typedef struct nodo* lista;
struct nodo
{
      char nombre[255];
      lista siguiente;
};
struct entradas_directorio
{
  lista entradas;
  int hubo_error;
  char descripcion_error[255];
};
struct cambiar_nombre
{
      char antiguo[512];
      char nuevo[512];
};
struct cadena
{
      int hubo_error;
      char descripcion_error[255];
      char nombre[512];
};
struct error
{
      int hubo_error;
      char descripcion_error[255];
};
```

Figura 1: Código

b) ¿Qué se puede generar a partir de este código?

a) El código de la figura 1 representa un manejo de sistema de archivos distribuido, como NFS. Los elementos son:

Structs:

- stat_archivo: representa los meta-datos del archivo, como los permisos asociados, propietarios, etc.
- archivo_escritura: representa los conjuntos de caracteres que contiene el archivo.
- archivo_lectura: representa la cantidad de caracteres a leer de un cierto archivo.
- archivo_atributos: representa los permisos/atributos/protecciones a modificar en un archivo.
- camino_t: representa el path/camino de un archivo.
- nodo: representa una lista de archivos/entradas.
- entradas_directorio: representa el contenido del directorio.
- cambiar_nombre: representa el nuevo nombre del archivo.

- cadena: representa un string de 512 caracteres.
- error: indica si hubo un error y su descripción.

Funciones:

- escribir: permite abrir un archivo para escritura y escribir datos en él.
- leer: permite leer una cierta cantidad de caracteres de un archivo.
- obtener_atributos: obtiene los meta-datos de un archivo.
- modificar_atributos: modifica los atributos/permisos de un determinado archivo.
- borrar_archivo: permite borrar un archivo.
- crear_archivo: crea un archivo.
- directorio_actual: retorna el nombre del directorio actual.
- cambiar_directorio: permite cambiar a otro directorio.
- crear_directorio: crea un nuevo directorio.
- listar_directorio: devuelve el contenido de un directorio.
- renombrar: cambia el nombre de un archivo o directorio.

b) Al ejecutar *rpcgen* con este programa, se generan:

- El stub del cliente.
- El stub del servidor.
- El header con los structs, funciones, etc.

También, se pueden generar:

- El código del cliente.
- El código del servidor.
- La definición de la interfaz XDR para los structs.
- Un Makefile.

8. Realizar un cliente que solicite a un servidor las siguientes consultas:

a) Su tiempo local. El formato que se debe observar es el siguiente: *Thu Oct 5 20:10:27 2011*.

b) El nombre de la máquina.

c) La cantidad de usuarios logueados.

Debe estar implementado con RPC.

El código fuente de dicho ejercicio se adjuntó a la entrega. La carpeta que lo contiene se llama *consultas*.