UDS

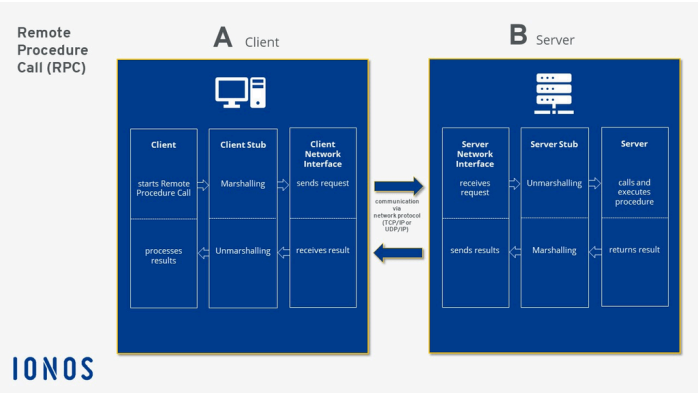
INGENERIA EN SISTEMAS COMPUTACIONALES

NOMBRE DEL ALUMNO:	Francisco López Argueta
NOMBRE DEL PROFESOR:	Andrés Alejandro Reyes Molina
MATERIA:	SISTEMAS OPERATIVOS DISTRIBUIDOS
ACTIVIDAD	SUPER NOTA DE LOS TEMAS INDICADOS

INTRODUCCIÓN. Es aquel que está gestionado por un servidor y sus clientes invocan sus métodos utilizando un "método de invocación remota". El cliente invoca el método mediante un mensaje al servidor que gestiona el objeto, se ejecuta el método del objeto en el servidor y el resultado se devuelve al cliente en otro mensaje.

1. RPC.

En computación distribuida, la llamada a procedimiento remoto (en inglés, Remote Procedure Call, RPC) es un programa que utiliza una computadora para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas. El protocolo que se utiliza para esta llamada es un gran avance sobre los sockets de Internetusados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando estas encapsuladas dentro de las RPC

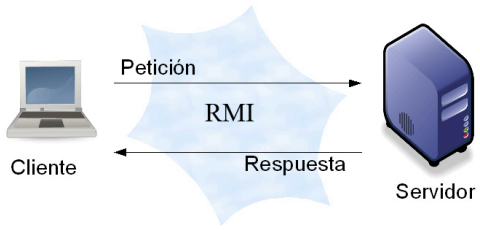


2. Ejecucion RPC en c#

RPC usa un enfoque de contrato primero para el desarrollo de API. Los búferes de protocolo (Protobuf) se usan de forma predeterminada como el lenguaje de definición de interfaz (IDL). El archivo *.proto contiene: La definición del servicio gRPC Los mensajes enviados entre clientes y servidores Para más información sobre la sintaxis de los archivos de protobuf, consulte Creación de mensajes de Protobuf para aplicaciones .NET. Vamos a considerar, por ejemplo, el archivo greeter.proto que se usa en Introducción a un servicio gRPC: Define un servicio Greeter. El servicio Greeter define una llamada a SayHello. SayHello envía un mensaje HelloRequest y recibe un mensaje HelloReply:

3. RMI

RMI (Java Remote Method Invocation) es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.



4. Ejecucion en Java de RMI.

Enviar mensajes no tiene gran utilidad, ¿no sería mejor poder llamar a algunas funciones en el servidor? esto es justamente lo que hace RMI en Java. Declaramos el servidor, el cliente, y la interfaz que servirá como “pegamento” entre estos dos y que se encargará del paso de parámetros. Después, desde el cliente llamamos a las funciones declaradas anteriormente. De esta manera, todos los métodos se ejecutarán en el servidor. En este caso haremos una calculadora, para no hacer el post muy largo, pero podemos hacer miles de cosas; se me ocurre, por ejemplo, conectar a una base de datos.

5. Diferencia entre RPC y RMI

RPC y RMI son los mecanismos que permiten a un cliente invocar el procedimiento o método desde el servidor a través del establecimiento de la comunicación entre el cliente y el servidor. La diferencia común entre RPC y RMI es que RPC solo admite la programación de procedimientos, mientras que RMI admite la programación orientada a objetos. Otra diferencia importante entre los dos es que los parámetros pasados a la llamada a procedimientos remotos consisten en estructuras de datos ordinarias. Por otro lado, los parámetros pasados al método remoto consisten en objetos.

Bases para la comparación	RPC	RMI
Ayudas	Programación procesal	Programación orientada a objetos
Parámetros	Las estructuras de datos ordinarios se pasan a procedimientos remotos.	Los objetos se pasan a métodos remotos.
Eficiencia	Más bajo que RMI	Más que RPC y con un enfoque de programación moderno (es decir, paradigmas orientados a objetos)

6. Soporte del sistema operativo.

El sistema operativo es básicamente un programa que controla los recursos del computador, proporciona servicios a los programadores y planifica la ejecución de otros programas. A partir de los µP de 16 bits, las CPU incorporan estructuras de apoyo a los sistemas operativos, por lo que resulta interesante una introducción a dos de las funciones básicas del SO que más inciden en la arquitectura de la CPU: la multiprogramación (o multitarea) y el control de memoria.

7. Sistemas de archivos distribuidos.

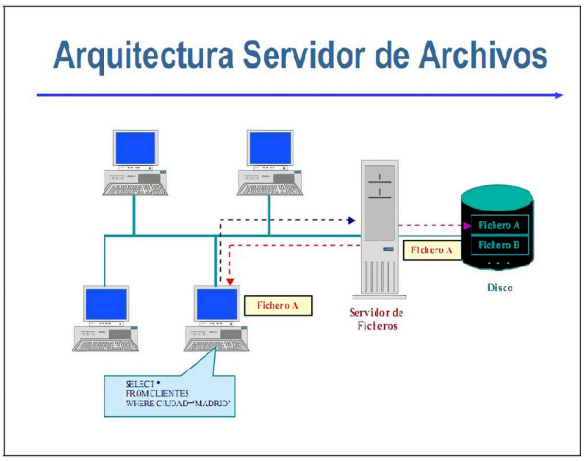
Un sistema de archivos distribuido nos va a permitir almacenar y acceder a archivos remotos como si fueran locales, sin que notemos pérdidas en el rendimiento. Vamos a analizar dos sistemas de archivos distribuidos muy conocidos (NFS y AFS), comentando sus principales ventajas e inconvenientes.

8. Características de los sistemas de archivos.

Tu disco duro externo, el disco duro interno de tu ordenador, un USB o una tarjeta SD. Todos ellos son unidades de almacenamiento, lo que quiere decir que cuando los formateas estás creando la infraestructura en las que van a alojarse los datos que después le quieras meter. Es aquí donde entra en juego el sistema de archivos, un componente del sistema operativo que se encarga de administras la memoria de cada unidad. Se encargan de asignarle a los archivos el espacio que necesiten, ordenarlos, permitir el acceso a ellos y administrar el espacio libre de las unidades de almacenamiento.

9. Arquitectura del servicio de archivo

La computación desde sus inicios ha sufrido muchos cambios, desde los grandes ordenadores que permitían realizar tareas en forma limitada y de uso un tanto exclusivo de organizaciones muy selectas, hasta los actuales ordenadores ya sean personales o portátiles que tienen las mismas e incluso mayores capacidades que los primeros y que están cada vez más introducidos en el quehacer cotidiano de una persona.

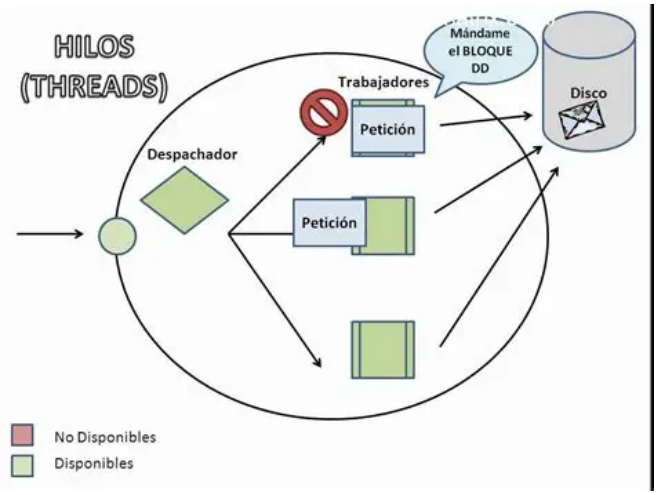


10. Procesos e hilos

En un sistema multiprogramado o de tiempo compartido, un proceso es la imagen en memoria de un programa, junto con la información relacionada con el estado de su ejecución. Un programa es una entidad pasiva, una lista de instrucciones; un proceso es una entidad activa, que empleando al programa– define la actuación que tendrá el sistema. En contraposición con proceso, en un sistema por lotes se habla de tareas. Una tarea requiere mucha menos estructura, típicamente basta con guardar la información relacionada con la contabilidad de los recursos empleados

11. Los hilos y el sistema operativo

Como se vio, la cantidad de información que el sistema operativo debe manejar acerca de cada proceso es bastante significativa. Si cada vez que el planificador elige qué proceso pasar de Listo a En ejecución debe considerar buena parte de dicha información, la simple transferencia de todo esto entre la memoria y el procesador podría llevar a un desperdicio burocrático2 de recursos. Una respuesta a esta problemática fue la de utilizar los hilos de ejecución, a veces conocidos como procesos ligeros (LWP, Lightweight processes). Cuando se consideran procesos basados en un modelo de hilos, se puede proyectar en sentido inverso que todo proceso es como un solo hilo de ejecución. Un sistema operativo que no ofreciera soporte expreso a los hilos los planificaría exactamente del mismo modo.



12. Patrones de trabajo con hilos

Un hilo tiene una tarea distinta de todos los demás: el hilo jefe genera o recopila tareas para realizar, las separa y se las entrega a los hilos trabajadores. Este modelo es el más común para procesos que implementan servidores (es el modelo clásico del servidor Web Apache) y para aplicaciones gráficas (GUI), en que hay una porción del programa (el hilo jefe) esperando a que ocurran eventos externos. El jefe realiza poco trabajo, se limita a invocar a los trabajadores para que hagan el trabajo de verdad; como mucho, puede llevar la contabilidad de los trabajos realizados. Típicamente, los hilos trabajadores realizan su operación, posiblemente notifican al jefe de su trabajo, y finalizan su ejecución.

13. Concurrencia

Desde un punto de vista formal, la concurrencia no se refiere a dos o más eventos que ocurren a la vez sino a dos o más eventos cuyo orden es no determinista, esto es, eventos acerca de los cuales no se puede predecir el orden relativo en que ocurrirán. Si bien dos procesos (o también dos hilos) completamente independientes entre sí ejecutándose simultáneamente son concurrentes, los temas que en la presente sección se expondrán se ocupan principalmente de procesos cuya ejecución está vinculada de alguna manera (p. ej.: dos procesos que comparten cierta información o que dependen uno del otro). Aunque una de las tareas principales de los sistemas operativos es dar a cada proceso la ilusión de que se está ejecutando en una computadora dedicada, de modo que el programador no tenga que pensar en la competencia por recursos, a veces un programa requiere interactuar con otros: parte del procesamiento puede depender de datos obtenidos en fuentes externas, y la cooperación con hilos o procesos externos es fundamental.

14. Mecanismos de sincronización

En la presente sección se enumeran los principales mecanismos que pueden emplearse para programar considerando a la concurrencia: candados, semáforos y variables de condición. Regiones de exlcusión mutua: candados o mutexes Una de las alternativas que suele ofrecer un lenguaje concurrente o sistema operativo para evitar la espera activa a la que obliga el algoritmo de Peterson (o similiares) se llama mutex o candado (lock). La palabra mutex nace de la frecuencia con que se habla de las regiones de exclusión mutua (mutual exclusion).

Es un mecanismo que asegura que cierta región del código será ejecutada como si fuera atómica. Sincronización: la exclusión de las secciones críticas entre varios procesos se protegen por medio de regiones de exclusión mutua. La figura ilustra varios procesos que requieren de una misma sección crítica; las flechas punteadas representan el tiempo que un proceso está a la espera de que otro libere el paso para dicha sección, y las punteadas verticales indican la transferencia del candado.