MCDI

Maestría en Ciencia de Datos e Información

INFOTEC POSGRADOS

Comunicación entre procesos

Dra. Magali Arellano Vázquez

INFOTEC

Problema de la sección crítica I

- Los procesos se pueden ejecutar concurrentemente
  - Puede ser interrumpido en cualquier momento, completando parcialmente ejecución.
- Acceso concurrente a datos compartidos puede dar lugar a incoherencias en los datos.
- Mantener la consistencia de datos requiere mecanismos para asegurar la ejecución ordenada de procesos cooperativos.

Condición de competencia o de carrera

La condición de carrera (race condition) ocurre cuando dos o más procesos acceden un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada. Suponga, por ejemplo, que dos clientes de un banco realizan cada uno una operación en cajeros diferentes al mismo tiempo.

Problema de la sección crítica II

Ejemplo: Condición de carrera

Alice quiere hacer un depósito. El Bob un retiro. Alice comienza la transacción y lee su saldo que es \$1000. En ese momento pierde su turno de ejecución (y su saldo queda como \$1000) y Bob inicia el retiro: lee el saldo que es \$1000, retira \$200 y almacena el nuevo saldo que es \$800 y termina. El turno de ejecución regresa a Alice la cual hace su depósito de \$100, quedando $\mathit{saldo} = \mathit{saldo} + 100 = 1000 + 100 = 1100$. Como se ve, el retiro se perdió y eso le encanta a los usuarios Alice y Bob, pero al banquero no le convino esta transacción. El error pudo ser al revés, quedando el saldo final en 800.

Otros problemas de concurrencia son: I

Postergación o Aplazamiento Indefinido Consiste en el hecho de que uno o varios procesos nunca reciban el suficiente tiempo de ejecución para terminar su tarea. Por ejemplo, que un proceso ocupe un recurso y lo marque como ocupado y que termine sin marcarlo como desocupado. Si algún otro proceso pide ese recurso, lo verá ocupado y esperará indefinidamente a que se desocupe.

Condición de Espera Circular Esto ocurre cuando dos o más procesos forman una cadena de espera que los involucra a todos. Por ejemplo, suponga que el proceso A tiene asignado el recurso cinta y el proceso B tiene asignado el recurso disco. En ese momento al proceso A se le ocurre pedir el recurso disco y al proceso B el recurso cinta. Ahí se forma una espera circular entre esos dos procesos que se puede evitar quitándole a la fuerza un recurso a cualquiera de los dos procesos.

Otros problemas de concurrencia son: II

Condición de No Apropiación Esta condición no resulta precisamente de la concurrencia, pero juega un papel importante en este ambiente. Esta condición especifica que si un proceso tiene asignado un recurso, dicho recurso no puede arrebatársele por ningún motivo, y estará disponible hasta que el proceso lo suelte por su voluntad.

Condición de Espera Ocupada (Busy waiting) Esta condición consiste en que un proceso pide un recurso que ya está asignado a otro proceso y la condición de no apropiación se debe cumplir. Entonces el proceso estará gastando el resto de su time slice revisando si el recurso fue liberado. Es decir, desperdicia su tiempo de ejecución en esperar. La solución más común a este problema consiste en que el sistema operativo se dé cuenta de esta situación y mande a una cola de espera al proceso, otorgándole inmediatamente el turno de ejecución a otro proceso.

Región crítica

Región crítica I

Región critica

- Considera un sistema de n procesos $p_0, p_1, \ldots, p_{n-1}$.
- Cada proceso tiene una sección crítica de segmento de código.
  - El proceso puede estar cambiando las variables comunes, actualizar la tabla, escritura de archivos, etc.
  - Cuando un proceso en el la sección crítica, ningún otro puede estar en su sección crítica.

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (true);
```

Región crítica II

- Problema de sección crítica es diseñar protocolo para resolver este.
- Cada proceso tiene que pedir permiso para entrar en su sección crítica en la sección de entrada, puede seguir con la sección crítica, sección de salida, entonces la sección restante.

Existen diferentes soluciones a este problema, como lo son:

Exclusión mutua Si el proceso P_i si el proceso P_i está ejecutando su sección crítica, ningún otro proceso puede estar ejecutando su sección crítica.

Progreso Si ningún proceso está ejecutando en su sección crítica y existen algunos procesos que desean entrar en su sección crítica, entonces la selección de los procesos que van a entrar en la sección crítica que viene no se puede posponer indefinidamente.

Región crítica III

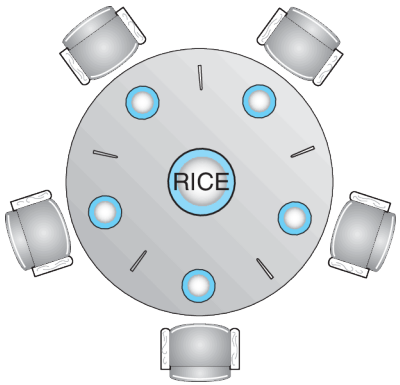
Espera limitada Un límite debe existir en el número de veces que otros procesos están autorizados a entrar en sus secciones críticas después de un proceso haya hecho una solicitud para entrar en su sección crítica y antes de que la solicitud se concede.

- Suponga que cada proceso se ejecuta a una velocidad distinta de cero.
- No hay supuestos concernientes a la velocidad relativa de los *n* procesos.

Los filósofos comensales I

El problema de los filósofos comensales

- Cinco filósofos se sientan alrededor de una mesa, frente a cada uno de ellos se encuentra un plato con arroz y un palillo a la izquierda.
- Cada uno de los filósofos comensales tienen que tomar dos palillos para poder comer.
- Se considera que sólo cuentan con un palillo a la izquierda habiendo sólo 5 palillos en total, indica que sólo pueden comer dos filósofos al mismo tiempo y los otros tres quedarían esperando.



Los filósofos comensales II

Condiciones: Deadlock y Starvation

El núcleo del problema consiste en resolver el hecho de que en algún momento los cinco filósofos comensales tomen su respectivo palillo de la izquierda pero queden esperando a que el comensal de al lado termine.
En este caso quedarían esperando para siempre ya que ningún filósofo soltaría su tenedor hasta haber comido (starvation) , lo cual tampoco sucederá porque los cinco filósofos comensales se encuentran en el mismo estado de interbloqueo (deadlock).

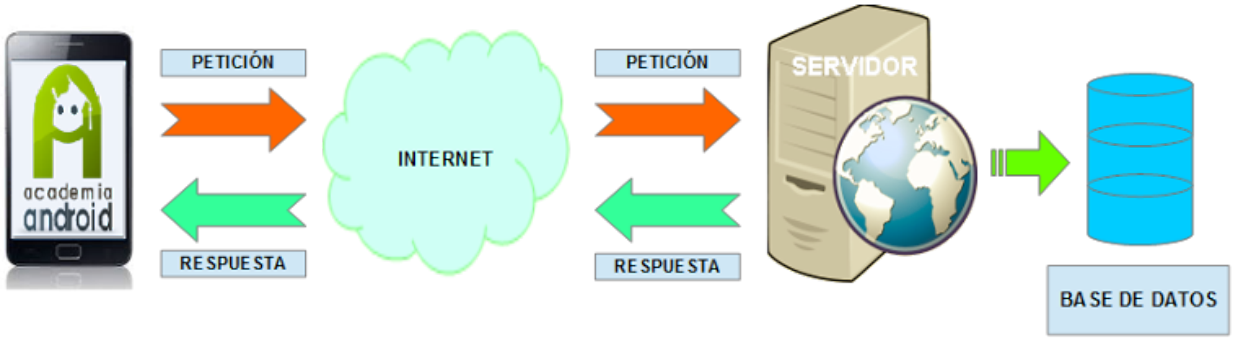
Deadlock Dos o más procesos esperan indefinidamente por un evento que puede ser causado solo por otro de los procesos que también está esperando.

Starvation Un proceso nunca ha sido removido de la lista de espera.

Comunicación

La comunicación entre procesos es el corazón de los sistemas distribuidos.

Modelo cliente servidor I



- El servidor puede o no estar en la misma máquina del cliente.
- Los servidores pueden dividir el conjunto de objetos en los que está basado el servicio y distribuirlos entre ellos mismos.
- Pueden mantener réplicas de los objetos en cada máquina.
- Un cache es un almacén de objetos de datos utilizados recientemente.
- Los caches pueden estar ubicados en los clientes o en un servidor Proxy que se puede compartir desde varios clientes.

Modelo cliente servidor II

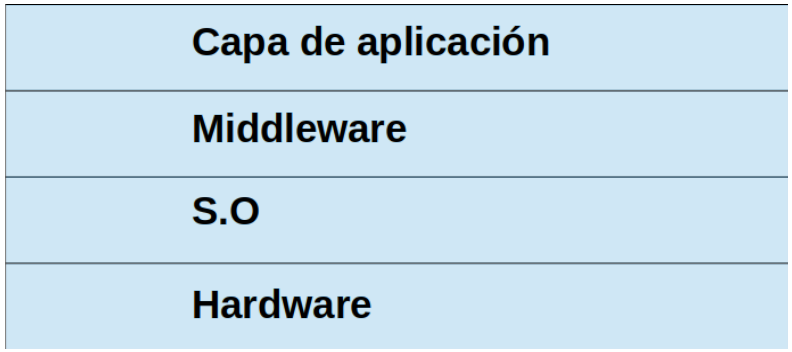
- El propósito de los servidores proxy es incrementar la disponibilidad y las prestaciones del servicio, reduciendo la carga en las redes de área amplia y en los servidores WEB.

Capas de software I

La arquitectura de un software se refiere a su estructuración como capas o módulos en un única computadora:

Plataforma El nivel de hardware y las capas más bajas del software.

Middleware Capa de software cuyo propósito es enmascarar la heterogeneidad y proporcionar un modelo de programación para los programadores de la aplicación.



Capas de software II

Ejemplos de Middleware:

- Sun RPC
- CORBA
- RMI
- DCOM
- RM-ODP

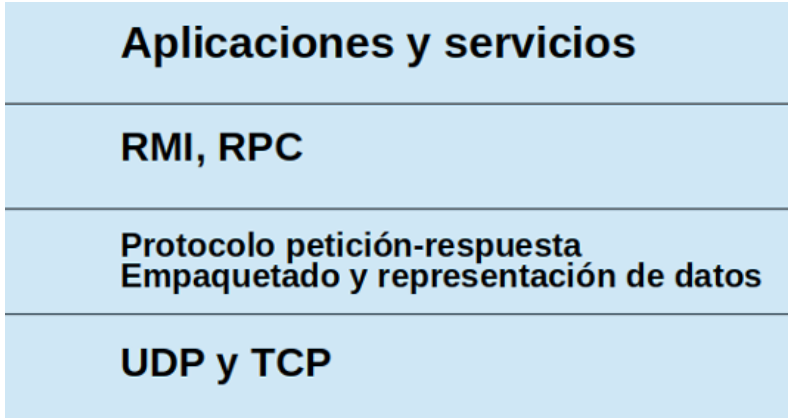


Figura: Middleware

Características:

- Comunicación Síncrona y Asíncrona
  - Existe una cola para cada destino de mensaje
  - Síncrona: el que envía y el que recibe son bloqueantes.
  - Asíncrona: el que envía no es bloqueante, el recibe puede serlo.
- Destinos de los mensajes.
  - Un puerto local es el destino de un mensaje dentro de un dispositivo.
  - Tiene un único receptor pero puede tener varios emisores.

Características:

- Fiabilidad.
  - En términos de validez.
  - Un servicio de mensajes es fiable si se garantiza que los mensajes se entregan.
  - Es no fiable si no se garantiza esto en términos de integridad.
  - Es fiable si los mensajes se entregan sin corromperse ni duplicarse.
- Ordenación.
  - Algunas aplicaciones necesitan que los mensajes sean entregados en orden.

Sockets I

- Los sockets (conectores) surgieron en UNIX BSD.
- Permiten una abstracción para la comunicación entre procesos.
- Cada puerto se asocia a un proceso, aunque cada proceso puede gestionar varios puertos.
- Un socket nos permite comunicarnos con otras computadoras.
- Existen tres dominios de comunicación para un socket:
  1. El dominio de UNIX (AF_UNIX): comunicación entre procesos del sistema..
  2. El dominio de Internet (AF_INET):comunicación a través de TCP(UDP)/IP. .
  3. El dominio NS.Comunicación de Xerox.

  NOTA: Python solo soporta los dos primeros tipos de comunicación.

Sockets II

- Tipo SOCK_DGRAM.- Sockets para comunicaciones en modo no conectado, con envío de datagramas de tamaño limitado. En dominios de Internet se usa el protocolo UDP.
- Tipo SOCK_STREAM.- Para comunicaciones fiables en modo conectado, de dos vías y con tamaño variable de los mensajes de datos. En dominios de Internet se ocupa el protocolo TCP.
- Tipo SOCK_RAW.- Permite el acceso a protocolos de más bajo nivel como el IP (nivel de red).
- Tipo SOCK_SEQPACKET.- Tiene las características del SOCK_STREAM pero además el tamaño de los mensajes es fijo.

Figura: Tipos de sockets

Sockets III

Socket	Descripcion
socket	Crea un socket del tipo y familia especificada.
socket.accept	Acepta nuevas conexiones.
socket.connect	Conecta el socket a la dirección dada en el puerto dado.
socket.send	Envia datos al socket especificado.
socket.recv	Recive información.

Figura: Funciones en python

RPC I

¿Qué es?

- Es un protocolo que permite a un programa ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.
- El cliente inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando éste de vuelta el resultado de dicha operación al cliente.
- XML-RPC es una implementación del protocolo RPC (Remote Procedure Call).
- En el mundo de los protocolos RPC se hizo un quiebre cuando aparecieron los llamados Web Services que no son otra cosa mas que la utilización de HTTP y XML como modo de transporte y codificación.
- Puede ser usado con Perl, Java, Python, C, C++, PHP y varios otros lenguajes de programación. Existen implementaciones para Unix, Windows y Macintosh.

RPC II

Para hacer una aplicación con XML-RPC en python necesitamos crear un código que se ejecute en el servidor y otro que se ejecute en el cliente.

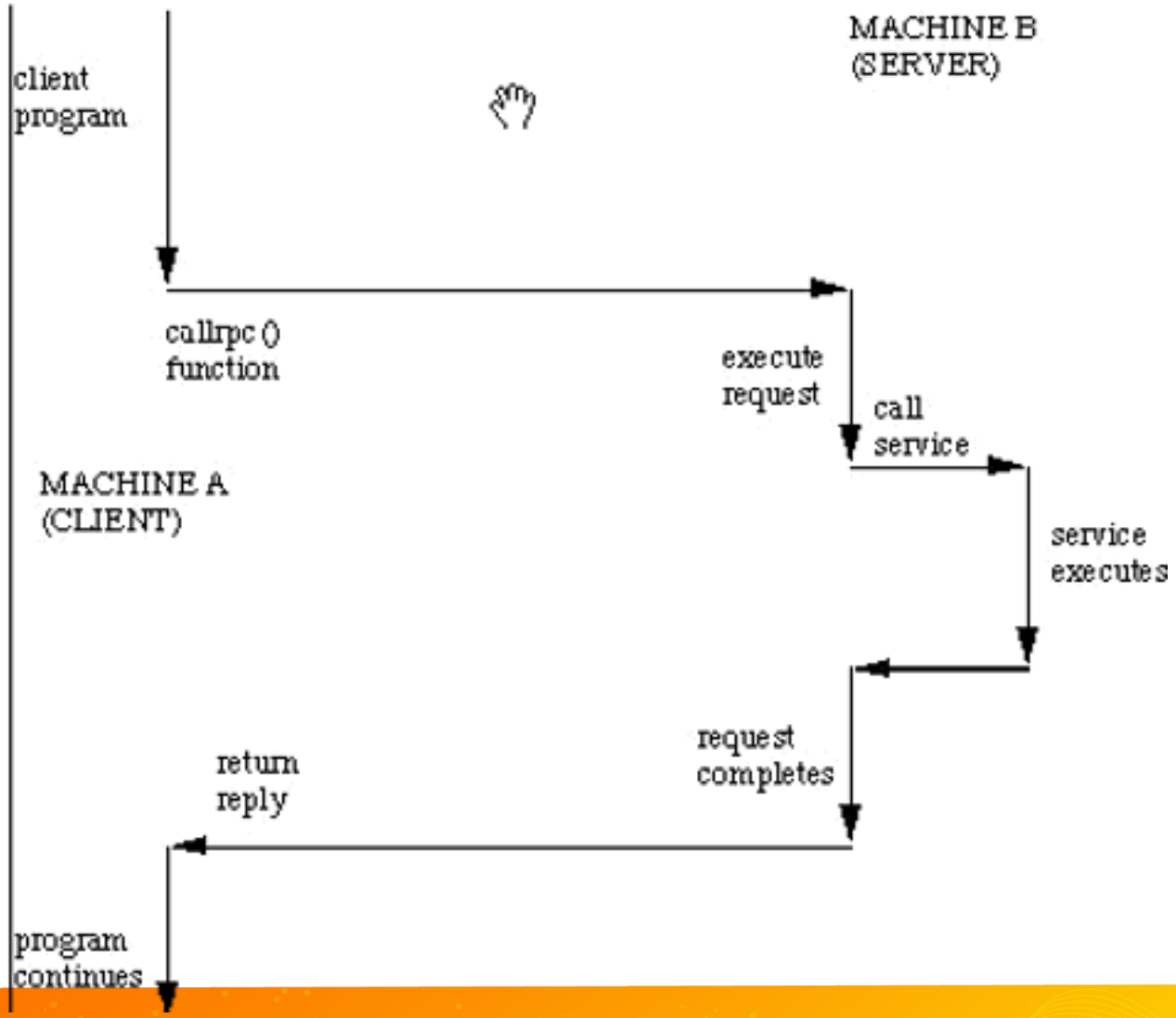
Servidor

- Se importa el módulo XMLRPC.
- Se declaran las funciones que se van a utilizar de manera remota.
- Se asocia a un puerto.
- Se registra la función y se inicia el servidor.

Cliente

- Se importa el módulo ServerProxy de xmlrpclib.
- Se conecta al equipo por el puerto asociado.
- Se llama a la función.

RPC III


client
program
MACHINE B
(SERVER)
callrpc ()
function
execute
request
call
service
MACHINE A
(CLIENT)
service
executes
request
completes
return
reply
program
continues