Resumen pa´l tercer parcial

Alfalfa producciones

SiSop

Índice

1. Comunicación entre procesos

La comunicación entre procesos cuenta con 2 primitivas: SEND y RECIVE. Estos pueden ser bloqueantes (sincrónicos) o no bloqueantes (asincrónicos). Características a tener en cuentas son: la conexión y la confiabilidad. El SEND puede tener un buffer.

Para ubicar un procedimiento remoto se puede tener un *binding* dinámico o uno estático.

2. Procedimientos remotos

- asincrónica: Cliente-Servidor
- sincrónica: RPC

2.1. RPC - Remote Procedure Call

1. El procedimiento cliente le envía (vía *stack*) en pedido a su *stub*
2. el *stub* cliente arma el mensaje y se lo envía al *kernel*
3. *kernel* cliente le envía el pedido al *kernel* del servidor
4. el *kernel* del servidor, se lo pasa a su *stub*
5. el *stub* del servidor, se lo pasa al procedimiento del servidor
6. el procedimiento del servidor ejecuta y después devuelve el resultado por el camino inverso

Las valores se pasan por *copy/restore*. Cualquier otra forma tiene problemas.

2.1.1. Fallas en RPC

- No se encuentran (servidor y cliente) ⇒ Excepción.
- Se pierde el requerimiento ⇒ Retransmición
- Se pierde el mensaje de vuelta ⇒ Retransmición
- Se cae el servidor antes de devolver la respuesta ⇒ Lo resuelve la política del servidor
- Se cae el servidor antes de procesar el pedido. Distintas políticas:
  - *at least once* operaciones idempotentes
  - *at most once*
  - *exactly once* muy dificil de lograr
- Se cae el cliente produciendo huerfanos ⇒

  Reencarnación Al levantarse hace *broadcast* avisando que volvió y quien era y cada servidor mata los procesos huerfanos

  Reencarnación suave Cuando llega un *broadcast* avisando que volvió una máquina, cada servidor verifica si sus computos remotos siguen teniendo dueño. ACA HAY ENIE.

  Expiración Exterminación el *stub* cliente lleva un *log* de los pedidos que se hicieron. Al levantarse mata explicitamente dichos procesos.

2.2. Sockets

Hay un *socket* con nombre en el servidor y uno sin nombre en el cliente.

El servidor crea un *socket* con nombre y espera una conexión.

"Un *socket* es un descriptor con varias cositas adentro"

Hay con conexión y sin conexión.

2.2.1. Socket con conexión

- El servidor abre el canal, publica su dirección y se pone a esuchar en busca de nuevas conexiones
- El cliente crea un socket y pide conectarse con el servidor
- El servidor acepta la conexión, entonces se lleva a cabo
- El cliente escribe el pedido
- El servidor lee el pedido, lo resuelve y escribe los dato al cliente
- El cliente lee los datos y cierra el canal

La comunicación ser realiza con WRITE y READ

2.2.2. Socket sin conexión

- El servidor abre el canal, publica su dirección y se bloquea esperando que llegue información
- El cliente crea un socket y pide conexión (ubicar su dirección en el *binding*) con el servidor y envía el pedido
- El servidor resuelve y envía los dato al cliente
- El cliente recibe los datos y cierra el canal

La comunicación ser realiza con SEND y RECEIVE

3. Diseño de un sistema operativo

3.1. Transaparencia

Locación los usuarios no deben saber donde están los recursos

Migración los recursos pueden cambiar de migración sin cambiar su nombre

Replica los usuarios no pueden decir la cantidad de copias existentes. i.e. varias copias de un archivo

Concurrencia varios usuarion pueden compartir recursos de manera automática. El SO asegura acceso secuencial no concurrente

Paralelismo Las actividades pueden ocurrir paralelamente sin que los usuarios lo sepan.

3.2. Flexibilidad

3.2.1. Kernel monolítico

Tiene la mayoría de los servicio. Centralizado, con capacidades red e integración de servicios remotos. Las llamadas al sistema se realizan mediante interrupciones

3.2.2. Microkernel

Provee lo mínimo indispensable, el grueso de los servicios los carga el sistema operativo. Proporciona 4 servicios:

- Un mecanismo de comunicación entre procesos
- Cierta administración de memoria
- Una cantidad limitada de planificación y administración de procesos de bajo nivel
- Entrada/Salidad de bajo nivel

3.3. Confiabilidad

Debe ser altamaente disponible, y debe cuidar bien los datos que se le dan. Además de ser seguro y tolerante a fallas.

3.4. Performance

- Tiempo de respuesta
- Rendimiento (medido en cantidad de trabajos por hora)
- Uso del sistema
- Cantidad consumida de la capacidad de red

3.5. Escalabilidad

NFS no es escalable.

4. TCP/IP - OSI/ISO

	TCP/IP		OSI/ISO
		7	Aplicación
5	Aplicación	6	Presentación
		5	Sesión
4	Transporte	4	Transporte
3	Internet	3	Red
2	Interfaz de red	2	Enlace de datos
1	Hardware	1	Física

Capas:

Física maneja bits

Enlace maneja paquetes, detecta errores de transmición por checksum.

Red se encarga del *ruteo* y asegura la trayectoria. La anterior sólo dirige a LAN, esta sirve para direcciones "'más grandes".

Transporte mantiene el orden de los paquetes para que tenga sentido el mensaje

Sesión es un refinado de la de transporte, sabe con quién es la comunicacióny ayuda a la sincronización

Presentación se encarga de homogeneizar los datos, facilitando la comunicación entre máquinas con representaciones distintas.

Aplicación interactúa con los usuario

5. Rendez-Vous

Asimétrica-Semi sincrónica

Semisincrónica: send no bloqueantes y receive bloqueante

6. Mailbox

Desacopla el emisor y receptor

Mayor flexibilidad en mensajes.

Desventaja: centraliza.

Modalidades:

- uno a uno
- muchos a uno (cliente servidor)
- uno a muchos (broadcast)
- muchos a muchos (comunicación entre servers)

Colas con prioridad.

7. Sincronización

7.1. Relojes

Se usan para la entrega de mensajes "a lo sumo una vez" y para mantener consistencia en las caches de los clientes.

Lógicos, cada proceso tiene uno propio. Lamport, los tiempos concuerdan con el orden de los procesos.

Físicos, hay varias versiones para sincronizar relojes de un sistema distribuido:

Cristian tiene un reloj universal en un servidor de tiempo (WWV), cada máquina se acomoda a ese tiempo

Berkeley cada máquina envía su tiempo a un servidor y este responde con la corrección que hay que realizar

Promedios todos hacen *broadcast* de su hora y a su vez, todos leen las horas de los demás, las promedian y se quedan con ese dato

Multiples fuentes externas parecido al anterior, salvo que se queda con la intersección entre todas las señales de los tiempos recibidos

7.2. Exclusión mutua

Hay varias versiones:

centralizado Existe un coordinador, su caida implica que nadie más entre a zonas de uso exlusivo. Se requieren 3 mensajes para el uso de una zona dedicada (pedido, ack, liberado).

distribuido (Ricart-Agrawala) requiere sincronización de los relojes lógicos. Un nodo pregunta a todos, y hasta que no obtiene todos los OK no entra en la zona crítica, si un nodo está usando no responde. Al liberar se avisa a todos. Empates se definen por menor marca temporal. Hay problema si se cae un nodo porque no responde. Se puede mejorar haciendo que todos contesten o que se acceda por mayoría simple. Tiene n puntos de falla. Todos los nodos tienen que tener una lista de procesos activos. Se requieren $2(n-1)$ mensajes para acceder en la primer versión.

token-ring red en forma de anillo, se pasa un token, el que tiene el token tiene permiso. Un problema es la perdida del token, porque se cae el que lo tenía o porque es lento. Requiere entre 1 y n mensajes

7.3. Recuperación de caidas del coordinador

Bully El que detecta que se cayó el coordinador manda un mensaje a cada nodo con número mayor a él. Si alguno responde, entonces, ese repite la operatoria hasta que alguno no reciba ninguna respuesta, se considera que ese es el de mayor número y asume como nuevo coordinador avisando a todos. Cuando se restaura el coordinador caido reasume su puesto.

Anillo Parecido al anterior, sólo que se pasa una lista por un anillo y se van agregando los nodos vivos. Al dar toda la vuelta el que inició este mecanismo, decide cuál es el mayor y ese es el que asume como nuevo coordinador.

8. Transacciones

Todo o nada.

Primitivas: atómicas (indivisible), consistente, serializable (las concurrentes no interfieren), permanentes (una vez realizado el commit los cambios son permanentes).

8.1. Implementación de las transacciones

- En espacio privado de trabajo, se copia todo, en caso de rollback se borra el espacio de trabajo, en caso de commit se reapuntan los punteros de los datos.
- Log de grabación adelantada. Graba el valor viejo y el nuevo. En el commit el log es el cuello de botella. Es más tolerante a caidas.

8.1.1. Subtransacciones

La transacción madre dispara transacciones hijas. El commit de las subtransacciones sólo es permanente en el universo de la transacción madre. En caso de que la transacción madre aborte o haga rollback los datos afectados por las subtransacciones no se ven fuera.

8.1.2. Two-phase commit

Cuando una transacción involucra varios procesos nodos, se tienen que poner todos "de acuerdo" para realizar el commit. Hay un coordinador, que en general suele ser el que inicia la transacción.

Todo se escribe en el log y se envían como *mensaje*

1. (INICIO FASE 1) El coordinador escribe "preparado" a todos los subordinados
2. Todos los subordinados, cuando lo están, envían "listo"
3. El coordinador revisa que todas las respuestas sean "listo" (FIN FASE 1)
4. (INICIO FASE 2) El coordinador escribe "commit"
5. Los subordinados escriben "commit" y lo realizan
6. Los subordinados envían "terminado" (FIN FASE 2)

Posibles problemas:

- Si se cae el coordinador luego de enviar "commit", el volver ve las respuestas en el log.
- Si se cae el subordinado ve la decisión en el log
- Si un subordinado no responde, se produce un *timeout* y se decide si cancela o no.
- si se cae el coordinador luego de escribir el log, al volver puede seguir.

8.2. Control de concurrencia

8.2.1. Bloqueo

Depende de la granularidad. Lectura (varios concurrentes). Escritura (sólo uno). Es costoso pero no está libre de deadlock.

Bloqueo de dos etapas

Reduce la posibilidad de deadlock. Hay una etapa de pedir recursos hasta que llega al punto de bloqueo (tiene todos los recursos). Puede usarse *hold&wait*, timeout y otros para resolver posibles bloqueos.

Si las transacciones usan esto son serializables.

8.2.2. Control optimista

Otorga todos los permisos, al momento de hacer el commit verifica si todo está como al iniciar la transacción, si algo cambió no se hace commit. Está libre de *deathlock*. Permite gran paralelismo. Anda muy bien con espacio privado de trabajo.

8.2.3. Sellos temporales

Cada transacción tiene un sello temporal para lectura y escritura. Cuando una transacción encuentra un sello mayor qe el suyo, aborta. Está libre de *deathlock* pero es dificil de implementar.

8.3. Deathlock

La detección en un sistema distribuido puede dar falsos anuncios por la demora de los mensajes.

Se puede hacer detección enviando un mensaje que si llega a volver al que lo envió este se suicida. El mensaje tiene un id de proceso original y los recursos que están involucrados.

8.3.1. Formas de prevención

1 recurso por vez

Asignación total

Pedido ordenado de recursos

Wound-Wait (espera y golpea) El viejo desaloja al nuevo. El joven espera al joven. Permite inanición.

Wait-Die (espera y golpea) El viejo espera un recurso que tiene un joven. El joven que quiere un recurso que tiene un viejo muere y debe empezar de nuevo (sin cambiar su sello temporal). Permite inanición.

9. Caches en los clientes

9.1. Sin reloj

Decide invalidar algo ??? Revisar

9.2. Con reloj

Al acceder se le da fecha de expiración. Si otro lo pide espera expiración o cancela permiso.

10. Filesystem

Modelos:

- Upload/Download
- Remote Access
- Transparencia de nombres

10.1. Semántica de archivos compartidos

Unix cambios instantaneos (cada 3 segundos se refresca)

Sesión se guardan los cambios al finalizar

Archivos inmutables sólo se permite crea y leer (tipo cintas)

Transacciones son de tipo sesión.

10.2. Tipos de servidor

Servidor con estado	Servidor sin estado
Permite lectura adelantada	No necesita open/close
Hay lock de archivos	Sin limite de cantidad de archivos abiertos
Mejor desempeño	Más tolerante a fallas
Fácil idempotencia	No desperdicia espacio en tablas
Solicitudes más cortas	Solicitudes más largas
Cuando se cae se pierde todo. La recuperación queda a cargo de los clientes	Si falla el cliente no se pierde nada. No importa
Si el cliente se cae el servidor no sabe como eliminar las entradas	Si se cae el cliente todo ok.

10.3. Caching

Distintos niveles. Problemas de consistencia.

Metodos:

Escritura a traves de cache funciona. No afecta el tráfico de escritura

Escritura demorada Mejor desempeño. Posible semántica ambigua.

Escritura al cierre Semántica de sesión.

Control centralizado Semántica Unix. No robusto. Poco escalable.

10.4. Implementaciones

- RFS - Remote File System
- NFS - Network File System
- AFS - Andrew File System

10.4.1. RFS

- Estado en el cliente y en el servidor
- Todo RPC
- Caida del servidor afecta al cliente
- Permite lecturas adelantadas
- Control de acceso adelantado
- Existe una cantidad máxima de archivos en operación (abiertos al mismo tiempo)

10.4.2. NFS (mount protocol)

- El servidor, si tiene el acceso permitido, devuelve un file handle.
- El file handle identifica el archivo que guarda. En Unix es un id y un nro. de i-nodo para identificar el directorio montaod dentro del FS exportado
- El servidor tiene una lista de los cliente y las cosas que montaron

Servido	Cliente
Exporta FS	Monta archivos y directorios
R/W según diga el cliente	R/W que pide al NFS que realice
Sin estado	Con estado
Comunicación por RPC	Comunicación por RPC
Sin Caché	Con Cache
Exporta FS y registra lo exportado y da un puntero al cliente	Con el handler tiene un inodo-v

10.4.3. AFS

- Ocultamiento (sesión)
- Espacio único de nombres
- El cliente se conecta por red Venus al servidor (red de servidores llamada Vice) pidiendo un archivo
- Cada cliente tiene un servidor, pero este último está conectado a otros servidores
- Entre Venus y Vice hay un identificador llamado fid que tiene el número de volumen, el número de vnodo y un código único