dit
UPM

SOLUCIÓN AL EXAMEN DE
SISTEMAS OPERATIVOS (DISTRIBUIDOS)
3 de Febrero de 2003
PRIMERA PARTE: sin libros ni apuntes, DURACIÓN: 60 minutos
Respete el espacio especificado para cada respuesta.

Pregunta 1

Compare el algoritmo de exclusión mutua centralizado con el de Ricart Agrawala adaptados para 1000 procesadores en una red ethernet de 100 Mbps.

Subpregunta 1.1 (*0'3 puntos*)

¿Que mejoras haría en cada algoritmo para esa red?

Al centralizado ninguna. En el de Ricart y Agrawala haría que la petición de recurso se hiciera por medio de un multienvío a todos los demás. Como la ausencia de respuesta puede deberse a estar en zona crítica, a estar entrando, o a pérdida de mensaje, por lo que los que tienen el recurso deberían responder siempre, lo quieran o no.

Subpregunta 1.2 (*0'4 puntos*)

En el caso de no haber ningún error en la transmisión de los mensajes, ¿cuantos mensajes circularían por la red por cada acceso a región crítica? ¿Cuál elegiría atendiendo sólo a la eficiencia?

En el caso del centralizado, 3 mensajes (petición, concesión, liberación). En el otro habría 1001 mensajes (una petición, 1000 concesiones). Por supuesto, elegiría el primero.

Subpregunta 1.3 (*0'2 puntos*)

¿Identifica algún problema adicional en alguno de ellos que lo haga prácticamente inutilizable en esta red?

Sí. La explosión de asentimientos en el de Ricart Agrawala.

Subpregunta 1.4 (*0'2 puntos*)

Si hubiera varios recursos distintos a controlar, ¿como mejoraría el algoritmo más eficiente para eliminar cuellos de botella?

Si hubiera varios recursos, pondría varios coordinadores, cada uno en un procesador, ubicándolos aplicando una función de hash*, por ejemplo. Si debe ser tolerante a fallos, cada máquina tendrá apuntado un coordinador por recurso, y en las elecciones, las prioridades dependerán del identificador del recurso.*

Subpregunta 1.5 (*0'6 puntos*)

¿Cuál elegiría atendiendo a la tolerancia a caidas de un procesador? Explique los casos posibles de fallo y el modo de resolverlos.

> *Elegiría también el centralizado ya que ammbos son igualmente intolerantes y requieren una recuperación..*
>
> *En el caso del centralizado puede caer un cliente o el coordinador. Si cae el cliente mientras ocupa un recurso, lo retendrá indefinidamente, a menos que el coordinador ponga plazos al mentenimiento de recursos ocupados (peligroso) o tenga un detector de fallos de los clientes, sondeando periódicamente a los que tienen un recurso. Si cae el coordinador, lo detectará uno o varios clientes y se iniciará una elección (el algoritmo del abusón puede valer, modificado para multienvío de las peticiones de coordinación). El coordinador elegido, antes de funcionar como tal, debe reconstruir el estado consultando a los clientes si están en zona crítica, están esperando, o no tienen interés.*
>
> *En el caso de Ricart y Agrawala todas las caidas impiden avanzar. Pueden utilizarse plazos o sondeos a los que no responden, para sacarlos fuera del grupo, reduciendo el número de interlocutores a los que pedir permiso..*

Subpregunta 1.6 (*0'2 puntos*)

¿Requiere alguno de estos algoritmos una sincronización de relojes físicos? ¿Para qué?

> *No. Sólo es necesario que las frecuencias no difieran mucho para que los plazos de detección de fallos sean comparables y controlados. La estabilidad de los relojes actuales hace innecesaria cualquier actuación en este sentido.*

Subpregunta 1.7 (*0'2 puntos*)

¿Requiere alguno de estos algoritmos mantenimiento de relojes lógicos? ¿De qué tipo y para qué?

> *Sí. El de Ricart y Agrawala necesita relojes de Lamport con resolución de empates con número de procesador para ordenar globalmente las peticiones.*

Pregunta 2 (*1'5 puntos*)

Se tiene un sistema distribuido formado por un proceso cliente y otro servidor que residen en nodos diferentes de una red. El servidor exporta una operación que el cliente invoca. y tanto la petición de la operación como la respuesta a la misma se realizan mediante mensajes.

Para este sistema se pide analizar sus posibilidades de recuperación frente a fallos esporádicos utilizando la simple técnica de poner plazos (*timeouts*) y hacer reintentos de la petición en el cliente. Se suponen dos modos posibles de construir el servidor, OISSE (Operación Idempotente y Servidor Sin Estado) y ONISCE (Operación No Idempotente y Servidor Con Estado), y cuatro tipos de fallos esporádicos simples, a saber:

1. Caída del servidor seguida de su pronta recuperación.
2. Caída del cliente seguida de su pronta recuperación.

3. Pérdida de un solo mensaje de petición en la red.

4. Pérdida de un solo mensaje de respuesta en la red.

Más concretamente, se pide escribir una tabla de dos columnas (OISSE y ONISCE) y cuatro filas, cada una de ellas para cada uno de los cuatro tipos de fallo en orden, colocando en cada casilla bien un **Sí**, o bien un **No**, según si la técnica de los plazos y reintentos permite recuperar del fallo, o no lo permite.

OISSE	*ONISCE*
Sí	*No*
Sí	*No*
Sí	*Sí*
Sí	*No*

Pregunta 3 (*1 punto*)

¿Cuál es la ventaja principal de la observación pasiva de una computación distribuida sobre la observación activa? ¿Cuál es el coste extra en el que se incurre?

Frente a la observación activa, la observación pasiva –también llamada reactiva– permite, a partir de una sola ejecución del sistema, construir el retículo de todos sus estados globales coherentes –no solo algunos de ellos, como la observación activa– y así captar toda la computación distribuida. De esta forma, la observación pasiva puede, por ejemplo, evaluar predicados no estables, algo que la observación activa no puede.

Con respecto a la observación activa, el coste extra en el que incurre la observación pasiva es un mayor número de mensajes, mensajes más largos –debido a la transmisión de las horas vectoriales–, mayor gasto de memoria –para almacenar los relojes vectoriales– y mayor trabajo de computación para realizar mensajería de observación causal, mientras que en el observador activo solo hace falta mensajería FIFO (si bien, conviene mencionar, en este caso ha de ser FIFO también la mensajería de computación, no solo la de observación).

Pregunta 4 (*1 punto*)

En la figura 2 del artículo de Babaoğlu y Marzullo visto en clase, y hablando de observaciones activas, se muestra el corte incoherente C'.

Se pide mostrar sobre esa misma figura una hipotética ejecución del algoritmo de la instantánea de Chandy y Lamport, que dé como resultado —aparte de los estados de los canales, que aquí no nos interesan— ese mismo estado global C' (existe más de una solución, pero solo se pide una de ellas). Se supone que el grafo de nodos (procesos) está completamente conectado.

Adicionalmente, se pide también demostrar sobre la figura cómo esa hipotética ejecución del algoritmo de la instantánea, en realidad no es posible.

Supongamos que el proceso p_1 inicia el algoritmo de la instantánea entre los sucesos e_1^3 y e_1^4, y que envía mensajes de instantánea en ese momento a los otros dos procesos, llegándole el suyo a p_2 entre los sucesos e_2^2 y e_2^3, y a p_3 después de e_3^6 (los mensajes de eco no interesan realmente, aparte de suponer que llegan más tarde que estos).

Pues bien, la situación descrita no es realmente posible, ya que el mensaje de instantánea entre p_1 y p_3 no puede llegar más tarde que el que p_1 envía a p_3 desde e_1^5, puesto que el algoritmo de la instantánea supone que la comunicación es FIFO.

Pregunta 5

En cierto sistema de memoria global compartida paginado y persistente se desea ordenar un fichero conteniendo un vector de 2^{30} flotantes de 64 bits cada uno. Disponemos para ello de 8 ordenadores de 2^{30} octetos de memoria libres cada uno (hay algo más de memoria donde caben justo el sistema operativo y nuestro programa). El tamaño de página es de 2^{12} octetos.

Para acelerar el cálculo, se decide usar los ocho procesadores. Inicialmente un proceso *proyecta* el fichero en memoria virtual, obteniendo su dirección virtual de comienzo[1], alineada al comienzo de una página. Seguidamente lanza 7 procesos adicionales, uno en cada procesador restanate, pasándoles a cada uno de ellos la dirección de un octavo del vector, para que ordene esa parte *in situ*, usando el algoritmo de la burbuja[2]. Él mismo se encarga de la porción restante y espera que los demás terminen.

Finalmente este proceso se encarga de mezclar (*merge*) las 8 porciones ordenadas. Para ello barre desde el principio al final las 8 porciones, eligiendo siempre el más pequeño de los examinados y escribiéndolo en el fichero del resultado, que también habrá sido proyectado en memoria virtual.

[1] La operación de proyección hace que un fichero se vea como una región de la memoria virtual, de modo que se pueda leer y escribir en él como si fuera memoria, utilizando el sistema de gestión de memoria virtual para que las operaciones se reflejen en disco. No carga ninguna página en memoria, por lo que el proceso que proyecta no se hace *propietario* de las páginas. En Unix una operación similar se llama `mmap`.

[2] Este algoritmo (*bubble sort*) barre repetidamente el vector, examinando pares consecuivos de números, que intercambia si están en orden incorrecto. El barrido se repite una y otra vez hasta que no se produzca ningún intercambio.

Subpregunta 5.1 (*0,4 puntos*)

¿Funcionaría mejor con invalidación al escribir, con actualización al escribir, o es indiferente? ¿Por qué?

> *Con memoria global soportada por paginación no es eficiente la actualización al escribir, debido al elevado coste de procesador. Sin embargo en este caso es indiferente bajo todos los puntos de vista, porque las porciones donde se hacen las ordenaciones parciales no son compartidas y por tanto no hay que actualizar ni invalidar, mientras que la operación de mezcla sólo lee.*

Subpregunta 5.2 (*0,4 puntos*)

Calcule, para el caso mejor, el número de faltas de página en cada procesador debidas exclusivamente a los datos a ordenar (no considere código ni variables auxiliares), diciendo a qué se deben.

> *Los procesadores auxiliares generarán* $2^{30}/2^{12} = 2^{18} = 262{,}144$ *faltas, para obtener sus datos. El primer proceso también, además de las faltas necesarias para traerse copias de las de los otros procesadores y las necesarias para escribir el resultado. En total* $2^{30}/2^{12} + 7 \times 2^{30}/2^{12} + 8 \times 2^{30}/2^{12} = 16 \times 2^{30}/2^{12} = 2^{34}/2^{12} = 2^{22} = 4{,}194{,}304$.

Subpregunta 5.3 (*0,4 puntos*)

¿Merece la pena la división del cálculo entre los 8 procesadores? ¿Por qué? Compárelo con el mismo algoritmo (ordenaciones parciales más mezcla) en un sólo procesador.

> *Sí, porque las 8 porciones que se ordenan por separado caben en memoria y se pueden aprovechar completamente los procesadores en paralelo durante las ordenaciones parciales, que son costosas, sin más faltas de página que las necesarias para traer los datos por primera vez. Si el coste de procesador fuera irrelevante, no merecería la pena.*

Subpregunta 5.4 (*0,4 puntos*)

¿Funcionaría el algoritmo si los procesadores tuvieran la mitad de memoria? ¿Y si tuvieran el doble? Si es así, explique las consecuencias sobre el rendimiento

> *El algoritmo sería igualmente correcto en ambos casos. Pero con la mitad de memoria, la porción del vector no cabe, por lo que cada barrido del algoritmo de la burbuja generará faltas de página. Con el doble de memoria no se mejorará nada.*

Subpregunta 5.5 (*0,4 puntos*)

¿Qué pasaría si la operación de proyección no diera una dirección alineada a comienzo de página?

> *Habría compartición falsa en algunas páginas, ocasionando invalidaciones innecesarias. Además se necesita una página más por procesador, que no cabe, y contrubuye aún más al trasiego.*

Subpregunta 5.6 (*0,4 puntos*)

¿Es necesaria la consistencia secuencial (*sequential consistency*) en este caso o valdrá algo más débil? Explique si haría falta algún punto especial de sincronización ligado a la memoria virtual.

En la primera fase del algoritmo no se comparte nada, por lo que no es necesario ningun mecanismo de consistencia. Sin embargo, en la segunda fase, es necesario leer secuencialmente todas las posiciones, que deben obtenerse actualizadas, por lo que el fin de la espera por la terminación de los procesos hijos debe garantizar que las páginas actualizadas por ellos estén accesibles. Como el proceso inicial no tenía copias de las páginas de los procesos auxiliares, bastan obtener copias de las páginas de los propietarios, que ya sabemos que han terminado. No es necesario pues ningún mecanismo de consistencia, salvo esa garantía, débil, de que una vez terminado un proceso, vamos a obtener las últimas modificaciones que produjo.

Pregunta 6

Se dispone de un servidor replicado según el modelo de la arquitectura de cuchicheo, con ordenación causal de las actualizaciones. El sistema está formado por tres gestores de réplicas (G_1, G_2 y G_3) y dos frontales (F_1 y F_2). En un instante dado (que se supone que es el tiempo $t = 0$), tanto los frontales como los gestores tienen el mismo vector de tiempo (VTValor en los gestores de réplicas): (2, 3, 5).

Se produce la siguiente secuencia de operaciones:

- $t = 1$: F_1 envía una operación de actualización a G_3.
- $t = 3$: F_1 envía una operación de consulta a G_2.
- $t = 5$: F_2 envía una operación de consulta a G_1.

En este sistema:

- Los mensajes tardan una unidad de tiempo en llegar a su destino y procesarse. Se supone que no hay fallos de comunicación.
- Los mensajes de cuchicheo se mandan cada 8 unidades de tiempo. El primero se manda en el instante $t = 8$.
- Cada gestor multienvía los mensajes de cuchicheo a todos los demás.

Se pide resolver las siguientes cuestiones, justificando la respuesta.

Subpregunta 6.1 (*0'5 puntos*)

¿En qué instante de tiempo F_1 recibe la respuesta a su consulta?

La respuesta se recibe en el instante $t = 10$. No se puede contestar la consulta inmediatamente por que el vector de marcas del servidor no es menor o igual que la del frontal. Por tanto, no puede contestar hasta que le lleguen las actualizaciones que le faltan. Ésto se produce en el instante $t = 9$, en el que se recibe el mensaje de cuchicheo del gestor G_3 y se procesa. Entonces, se puede enviar la consulta a G_2.

Subpregunta 6.2 (*0'5 puntos*)

¿En qué instante de tiempo F_2 recibe la respuesta a su consulta?

La respuesta la recibe en el instante $t = 7$. Una unidad de tiempo para enviar el mensaje y procesarse la respuesta y otra en recibirse. La consulta se resuelve inmediatamente por que el vector de tiempo del frontal es menor o igual que el del gestor.

Subpregunta 6.3 (*1 punto*)

Mostrar para cada unidad del sistema (frontales y gestores) el contenido de sus vectores de tiempo de valor, en cada unidad de tiempo. Usar el esquema de tabla siguiente para su representación.

Tiempo	G_1	G_2	G_3	F_1	F_1	Razón del cambio

Tiempo	G_1	G_2	G_3	F_1	F_1	*Razón del cambio*
2	*(3, 5, 2)*	*(3, 5, 2)*	*(3, 5,* ***3****)*	*(3, 5, 2)*	*(3, 5, 2)*	*G_3 recibe un mensaje con una actualización de F_1, que se aplica, ya que es estable.*
3	*(3, 5, 2)*	*(3, 5, 2)*	*(3, 5, 3)*	*(3, 5,* ***3****)*	*(3, 5, 2)*	*F_1 recibe la confirmación de G_3*
9	*(3, 5,* ***3****)*	*(3, 5,* ***3****)*	*(3, 5, 3)*	*(3, 5,* ***3****)*	*(3, 5, 2)*	*G_1 y G_2 modifican su vector, después de recibir el mensaje de cuchicheo de G_3 con una actualización.*