dit
UPM

EXAMEN FINAL EXTRAORDINARIO DE SISTEMAS OPERATIVOS

12 de Junio de 2000

PRIMERA PARTE: sin libros ni apuntes, DURACIÓN: 50 minutos

Conteste en el espacio reservado para ello

Apellidos:	Nombre:

Pregunta 1 (*1 punto*)

Explicad cómo se puede utilizar la técnica de las regiones para implementar pilas (*stacks*) de tamaño máximo a priori desconocido.

También, explicad cómo esta misma técnica, junto con la memoria virtual, permite implementar pilas cuyo tamaño total puede ser muy superior al de la memoria física disponible para ellas (utilizar argumentos de "vecindad" o *locality*).

Basta con asignar una región por cada pila, con huecos de memoria lógica de gran tamaño entre regiones contiguas. Además, cada región tiene determinado uno de los extremos como "extensible" y, si la pila correspondiente se desborda por ese lado, el SO lo detecta y amplía automáticamente en determinado espacio (lógico) esa región.
*Este esquema, unido a la memoria virtual, permite gestionar pilas cuyo tamaño total sea mucho (ísimo) mayor que el de la memoria física que ocupan en cada momento. Y ello debido a la excelente "vecindad" (*locality*) que tienen las pilas, determinada por la forma cómo se acceden, solo por uno de sus dos extremos.*

Pregunta 2 (*1 punto*)

Explicad qué quiere decir que las operaciones en un sistema distribuido de ficheros sean "sin estado" (o idempotentes) y decir por qué esa característica es útil.

Se dice de una operación que es idempotente cuando el resultado de ejecutarla (total o parcialmente) varias veces, seguido de una ejecución completa, es equivalente a haberla ejecutado una sola vez. Ejemplos de operaciones idempotentes son la lectura y la escritura de un bloque de un disco y, de no idempotente, una transferencia bancaria entre dos cuentas. El concepto de operación idempotente es importante porque permite implementar semántica de exactamente una vez *en presencia de fallos (tanto del sistema de comunicación como del servidor) de manera secilla. Basta con realizar un esquema de reintentos con plazos (*time-outs*) hasta que se obtenga la confirmación del servidor de que la operación se ha realizado.*

Pregunta 3 (*1 punto*)

¿Un multienvío totalmente ordenado es causal? Justifíquelo.

No necesariamente. Orden total significa que las entregas se hacen en el mismo orden en todas partes, pero ese orden no tiene por qué ser consistente con la causalidad.

Pregunta 4 (*1 punto*)

Explique cómo se pueden acelerar las escrituras de NFS (o cualquier sistema de ficheros distribuido sin estado) con una memoria RAM no volátil. Relaciónelo con la *transparencia frente a fallos* (*failure transparency*).

Con una RAM no volátil no es necesario que las operaciones de escritura remota esperen a que la escritura correspondiente en disco termine, antes de retornar, ya que la copia en memoria se conserva durante las caidas. Si se pudieran perder los datos de una escritura confirmada por causa de una caída, no habría transparencia frente a fallos.

Pregunta 5 (*1 punto*)

¿Para qué se utilizan relojes lógicos en el algoritmo de Ricart y Agrawala de exclusión mutua distribuida?

Para que todos los interlocutores estén de acuerdo en el orden de las peticiones del recurso compartido. En particular, un proceso que quiere el recurso sólo lo toma si él lo pidió primero*).*

dit UPM

EXAMEN FINAL EXTRAORDINARIO DE SISTEMAS OPERATIVOS

12 de Junio de 2000

SEGUNDA PARTE: pueden utilizarse libros y apuntes, DURACIÓN: 75 minutos

Conteste en el espacio reservado para ello

Apellidos:	Nombre:

Pregunta 6

Para poder realizar entrega (potencialmente) causal de mensajes uno a uno (*unicast*), se diseña un sistema de relojes matriciales de dos dimensiones en el que los únicos sucesos de interés son los de envío de mensajes, con cada elemento $[i, j]$ de la matriz significando el número de mensajes enviados desde el proceso i al proceso j. Véase la figura que viene a continuación:

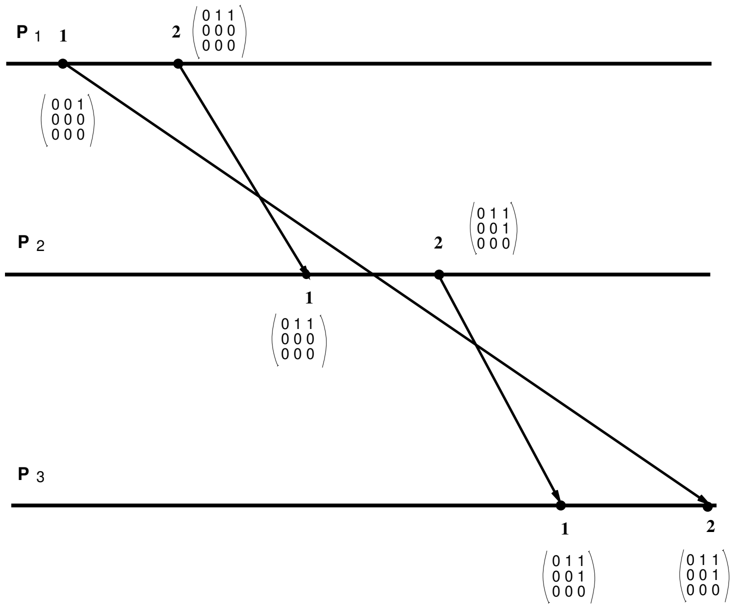

Figura 1: Horas matriciales en todos los sucesos de envío y recepción

Cuando un proceso P_i envía un mensaje a otro proceso P_j, se incrementa en una unidad la componente $MC_i[i, j]$, siendo MC_i el reloj matricial de P_i. El valor actualizado del reloj matricial de P_i se incluye con el mensaje en sí y, al entregarse éste en el proceso P_j, la hora matricial de P_j se actualiza según la regla:

$$(\forall m, n)\ MC_j[m, n] := max(MC_j[m, n], TS[m, n])$$

siendo TS la hora matricial que viene con el mensaje.

Está claro que en la figura mostrada, la mensajería no es causal. Sin embargo, se puede conseguir causalidad aplicando la siguiente regla de entrega:

Con cada proceso P_i asociamos un vector (columna) D_i que cuenta los mensajes provenientes de otros procesos ya *Delivered* (entregados) al proceso P_i. Cuando llega al proceso P_i un mensaje con hora matricial TS, se entrega éste *sii* la columna i-ésima de TS tiene un sólo elemento mayor que el correspondiente de D_i, y es mayor sólo en una unidad. Formalmente:

$$(\exists k \text{ tal que } TS[k, i] = D_i[k] + 1) \wedge ((j \neq k)\ TS[j, i] \leq D_i[j])$$

Si la regla anterior no se cumple, se almacena el mensaje correspondiente hasta que, debido a la entrega de otros mensajes, la regla se cumpla para dicho mensaje, en cuyo momento se entrega éste. Se pide,

Subpregunta 6.1 (*1 punto*)

Explicar cómo mediante la regla anterior se puede corregir la violación de causalidad que muestra la figura.

Al llegar al proceso P_3 el mensaje del proceso P_1, se ve que éste trae una hora matricial en cuya tercera columna hay más de un elemento (dos) que superan (en una unidad) a sus correspondientes en el reloj matricial local del proceso P_3 y, como resultado, el mensaje no se entrega.

Al llegar al proceso P_3 el mensaje del proceso P_1, sin embargo, exactamente un elemento de la tercera columna de la matriz que trae ese mensaje supera, y exactamente en una unidad, a su correspondiente en la tercera columna del reloj matricial local. Por ello, este mensaje se entrega al proceso P_3 y, al entregarlo y actualizarse apropiadamente el reloj local del proceso P_3, ya se cumple la regla de entrega para el mensaje del proceso P_2 al proceso P_3.

Así pues, los dos mensajes para el proceso P_3 se han entregado en este en orden causal.

Subpregunta 6.2 (*1 punto*)

Como las matrices pueden llegar a ser muy grandes en cuanto el número de procesos sube, y sin embargo éstas pueden codificarse de manera eficiente si tienen muchos elementos a cero, hay quien propone que, justo después de que un proceso P_i envíe un mensaje a un proceso P_j, P_i, ponga a cero todos los elementos de la columna j de su reloj matricial, con excepción del elemento $[i, j]$, y manteniendo el resto del algoritmo invariable. Se pide analizar esta propuesta y argumentar su validez o, en el caso de que no sea tal, aportar un contraejemplo (un ejemplo donde falle).

La propuesta es correcta. Para verlo basta con darse cuenta de que todos los mensajes $(m_{ij}^k | k > 1)$ que envíe un proceso P_i a un proceso P_j, y que sean posteriores al envío del mensaje m_{ij}^1 (justo después de enviar el cuál se pusieron a cero todos los elementos de la columna j en el reloj matricial de P_i, con excepción del elemento $[i, j]$), tendrán a m_{ij}^1 como antecesor (ya que precisamente el elemento $[i, j]$ no ha sido puesto a cero). Y como m_{ij}^1 ya tiene como antecesores a todos los mensajes que estaban representados por los elementos borrados, estos mensajes también serán tenidos en cuenta, por transitividad y de manera automática, como antecesores causales de todos los $(m_{ij}^k | k > 1)$. Por ello, no resulta necesario tener en cuenta explícitamente los elementos correspondientes de la columna j en el reloj matricial de P_i, y pueden ser puestos a cero.

Subpregunta 6.3 (*1 punto*)
En la misma línea de hacer que la mayor parte de los elementos de las matrices sean nulos, otra propuesta sugiere que los procesos tengan siempre la columna correspondiente a su propio índice en su reloj matricial, con todos los elementos a cero, también con el resto del algoritmo invariable. Se pide analizar esta segunda propuesta en el mismo sentido que la anterior.

La propuesta es correcta también. Pues los valores borrados solo representan mensajes que han sido antecedentes de otro(s) que ya ha(n) sido entregado(s) en el proceso destinatario —de hecho, de no hacer esto, siempre se tiene que, para todo proceso, la columna correspondiente a su mismo índice es idéntica a su vector D—, Es decir, mensajes que han sido ya tenidos en cuenta y entregados en orden causal en todo caso, por lo que las relaciones de precedencia causal que representan los valores a borrar, no es necesario tenerlas en cuenta más.

Pregunta 7
Un cliente intenta sincronizarse con un servidor de tiempos. Para ello envía sucesivamente cinco consultas con su fecha y hora (milisegundos) y espera respuestas durante un rato. A las 946080003400 de su reloj decide no esperar más y ajustar su reloj inmediatamente, habiendo recopilado hasta entonces cuatro respuestas, con dos marcas de tiempo cada una (hora de cliente y hora del servidor), según la tabla siguiente:

Hora del cliente	Mensaje recibido
946080003360	946080003342 — 946080003360
946080003363	946080003333 — 946080003351
946080003372	946080003350 — 946080003370
946080003385	946080003365 — 946080003387

Subpregunta 7.1 (*0.5 puntos*)
¿A qué valor lo haría?

Elegiremos la respuesta de menor retardo, al ser la de menor error. La corrección a aplicar es la diferencia entre la hora del servidor y la del cliente, mas la mitad del retardo de ida y vuelta. Es decir, que sumaremos 9 milisegundos al reloj local, obteniendo 946080003409, según se deduce de los cálculos siguientes:

Hora del cliente	*Mensaje recibido*	*Retardo de ida y vuelta*	*Corrección a aplicar*
946080003360	*946080003342 — 946080003360*	*18*	*9*
946080003363	*946080003333 — 946080003351*	*30*	*4*
946080003372	*946080003350 — 946080003370*	*22*	*9*
946080003385	*946080003365 — 946080003387*	*20*	*12*

Subpregunta 7.2 (*0.25 puntos*)
¿Qué error máximo cometería con los datos que tenemos?

El error máximo, por arriba o por abajo, es la mitad del retardo de ida y vuelta, es decir, 9 milisegundos.

Subpregunta 7.3 (*0.25 puntos*)
¿Qué cambiaría de sus respuestas anteriores si un mensaje tarda como mínimo 8 milisegundos en llegar a su destino?

No cambiaría el ajuste elegido, pero ahora al error habría que restarle el retardo mínimo, por lo que se queda en un milisegundo.

Pregunta 8 (*1 punto*)
Se desea sumar un vector de 10240000 números de coma flotante de 64 bits en un multicomputador de memoria virtual distribuida formado por 10 computadores con 20 megaoctetos de memoria cada uno. Los números están contenidos en un fichero que se puede proyectar directamente en memoria virtual. ¿Cómo organizaría el cálculo para obtener la mejor eficiencia? Justifíquelo:

1. Haría todo el cálculo con uno de los procesadores.

2. Utilizaría todos los procesadores, dividiendo el vector en 10 trozos contiguos, sumando cada uno un trozo y poniendo el resultado en una variable de una página compartida. Uno de ellos, cuando termine, espera que terminen los demás y suma sus resultados.

3. Lo mismo que en el caso anterior, pero el primer procesador se encargaría del primer número, del 11º, del 21º, del 31º, etc, el segundo procesador del 2º, del 12º, del 22º, del 32º, etc, y así sucesivamente.

4. Otra solución.

5. No es posible hacerlo.

Las soluciones 1 y 2 tiene el mínimo número de transferencias de página posibles. Sin embargo la 2 puede tardar unas 10 veces menos. La última necesita todas las páginas en todos los procesadores.