dit
UPM

EXAMEN FINAL ORDINARIO DE SISTEMAS OPERATIVOS DISTRIBUIDOS

4 de Febrero de 2000

PRIMERA PARTE: sin libros ni apuntes, DURACIÓN: 30 minutos

Conteste en el espacio reservado para ello

Apellidos:	Nombre:

Pregunta 1 (*1 punto*)
NFS, o cualquier otro sistema de ficheros fiable sin estado, es mucho más ineficiente en escrituras que en lecturas. ¿Por qué? ¿Como optimizaría un servidor de NFS para que las escrituras sean tan eficientes como las lecturas? Use una solución *hardware*.

Las escrituras son más ineficientes porque se quiere garantizar la semántica de al menos una vez*, incluso si se cae el servidor. Para ello, cuando la operación retorne, debemos tener garantía que los datos están en disco, o van a estarlo. Para ello normalmente se escribe en disco antes de retornar. Se puede optimizar haciendo persistente la memoria donde se alojen las peticiones de escritura; por ejemplo, poniéndole una batería que sea capaz de avisar cuando se agota y dé tiempo a volcar a disco las escrituras confirmadas y no realizadas.*

Pregunta 2 (*1 punto*)
¿Cuál es la diferencia fundamental entre los relojes lógicos de Lamport y los relojes vectoriales? Dad una ventaja concreta que puede obtenerse usando los segundos en vez de los primeros.

La diferencia fundamental es que mientras que los relojes lógicos solo suponen un morfismo de la relación de causalidad (es decir, $e \longrightarrow e' \Rightarrow (LC(e)) < LC(e'))$, pero la implicación inversa no es cierta), los relojes vectoriales son un isomorfismo de la relación de causalidad y, por ello, caracterizan completamente la computación distribuida.
Los relojes vectoriales suponen una ventaja sobre los lógicos a la hora de, por ejemplo, realizar observaciones coherentes eficaces de computaciones distribuidas. Pues permiten no solo evitar situaciones de hambruna que pueden aparecer con los relojes lógicos, sino, algo quizás más importante desde el punto de vista práctico, reducir los tiempos de espera y los tamaños medios de las colas de mensajes no estables (mensajes recibidos y aún no entregados, en espera de otros que pueden ser (o son) anteriores en el orden causal).

Pregunta 3 (*1 punto*)
Dar la estructura básica de las “credenciales” (*capabilities*) y explicar por qué en sistemas distribuidos, al contrario de lo que ocurre en sistemas centralizados, no es posible vivir sin ellas.

Una credencial está formada por una identificación del recurso al que se refiere, un conjunto cifrado (y, a veces, también sin cifrar) de los derechos de acceso que contiene sobre el recurso, y un número aleatorio grande, que es por ello (prácticamente) imposible de adivinar.
A diferencia de lo que ocurre en sistemas centralizados, en los que el kernel tiene conocimiento completo de un proceso cuando éste le realiza una petición, en sistemas distribuidos las peticiones vienen en general a través de la red con lo que, incluso cuando el peticionario es un kernel remoto, es en general no fiable. Por ello, las credenciales vistas (u otro método análogo) se hacen imprescindibles en estos sistemas.

Pregunta 4 (*1 punto*)
¿Plantean algún problema los fallos de página a un planificador de hebras realizado en el nivel de usuario?. Explicad brevemente.

Se presenta un problema similar al que ocurre cuando se realiza una operación de E/S. Pues, de nuevo, al no saber el kernel que el proceso es multihebra, le detiene completamente hasta que se restaure el fallo de página. Cuando, si en su lugar informara de este hecho al planificador en el nivel de usuario, éste bien podría planificar otra hebra del mismo proceso para ejecutar mientras se sirve el fallo de página, y así no se detendría el proceso completo.

Pregunta 5 (*1 punto*)
Explique qué tipo de multienvío es necesario para hacer replicación con réplicas activas (o máquinas de estado, o modelo totalmente síncrono), de modo que todas ellas pasen exactamente por los mismos estados.

Atómico totalmente ordenado, ya que para que todas las réplicas pasen exactamente por los mismos estados, deben recibir los mensajes en el mismo orden, y recibirlos todos o ninguno.

dit
UPM

EXAMEN FINAL ORDINARIO DE SISTEMAS OPERATIVOS DISTRIBUIDOS

4 de Febrero de 2000

SEGUNDA PARTE: pueden utilizarse libros y apuntes, DURACIÓN: 100 minutos

Conteste en el espacio reservado para ello

Apellidos:	Nombre:

Pregunta 6 (*2 puntos*)

Modificar el (seudo) programa de la figura 12 del artíclo de O. Babaoglu y K. Marzullo para que calcule y devuelva también el estado de los canales de comunicación, no sólo el de los procesos.

En el apartado de declaraciones, se añaden:

var *chi: array[1..n]* **of** *set* **init** *empty;*
  state: array[1..n] **of** *Boolean* **init** *false;*
  taking_snapshot: array[1..n] **of** *Boolean* **init** *false;*

En las dos alternativas `request` *y* `response` *se añade al principio:*

if *taking_snapshot[j]* **then** *chi[j] := chi[j] + m;*

Y la alternativa `snapshot` *se cambia a:*

snapshot:
  if *s = 0* **then**
    % this is the first snapshot message
    state := blocking;
    for *k:= 1* **to** *n* **do** *taking_snapshot[k] := true;*
    for *k:= 1* **to** *n* **do** *chi[k] := empty;*
    send *[type: snapshot]* **to** *p(1),...,p(i-1),p(i+1),...,p(n)*
  taking_snapshot[j] := false;
  s := (s + 1) mod n;
  if *s = 0* **then**
    % this is the last snapshot message
    send *[type: snapshot, data: (state, chi)]* **to** *p(0);*

Pregunta 7

El siguiente programa paralelo no debería imprimir nunca nada en un multiprocesador [1]:

```
a:= 0;
b:= 0;
cobegin
   for i:=1 to 1000000 do begin
                              br:= b;
                              ar:= a;
                              if (ar < br) then imprime("Error");
                            end;
   for j:=1 to 1000000 do begin
                              a:= a+1;
                              b:= b+1;
                            end;
```

[1] `cobegin/coend` delimitan procesos concurrentes, representado cada uno por una sentencia (simple o compuesta). `cobegin` arranca las sentencias en paralelo y `coend` espera a que todas terminen.

```
coend;
```

Sin embargo, si se ejecuta en un sistema distribuido con memoria virtual compartida, puede que en algunos casos se impriman mensajes de error.

Subpregunta 7.1 (*1 punto*)

Para cada modelo de consistencia diga si puede dar algún mensaje de error, justifíquelo muy brevemente y, si en algún caso la respuesta depende de algo no explícito en el programa, diga de qué:

1. Consistencia secuencial.

   No, porque todo entrelazado de operaciones de lectura y escritura de a y b dan que $a \geq b$.

2. Consistencia causal.

   No, si todas las escrituras en un proceso dependen causalmente. Ello ocasionará que las lecturas de b reflejarán el incremento anterior de a. Si no están relacionadas causalmente, puede haber errores.

3. Consistencia PRAM.

   No, porque las escrituras se propagan en orden FIFO.

4. Consistencia cache o coherencia de memoria.

   Si, porque sólo se ven en el mismo orden las escrituras en la misma variable.

5. Consistencia de procesador.

   No porque es PRAM.

6. Consistencia débil.

   Si, a menos que a y b estén marcadas como variables de sincronizacion.

7. Consistencia de liberación.

   Sí, porque no hay regiones críticas marcadas.

8. Consistencia de entrada.

   Sí, porque no hay regiones críticas marcadas.

Subpregunta 7.2 (*1 punto*)

En el caso de realizar al menos consistencia secuencial, ¿en qué caso podría haber más trasiego de páginas (*thrashing*) y en qué caso menos? Intente ordenarlos por trasiego y justifíquelo:

1. Replicando en lectura e invalidando al escribir.

   Cada escritura requiere una invalidación confirmada y cada lectura puede requerir una petición de página. El trasiego depende de las velocidades relativas de los procesos. Pueden llegar a producirse dos mensajes confirmados por escritura. Si el escritor mantiene la propiedad durante algún tiempo, el trasiego se reduce notablemente.

2. Replicando en lectura y actualizando al escribir.

   Cada escritura requiere una actualización remota. El trasiego es similar al último caso.

3. Sin replicar y manteniendo a y b en el primer proceso.

   El segundo debe pedir dato y actualizarlo remotamente para cada modificación. Se producen dos mensajes confirmados por escritura. Es el caso en que más datos se trasiegan.

4. Sin replicar y manteniendo a y b en el segundo proceso.

   El primero debe pedir el dato remotamente para cada lectura. Es el caso más corto, junto con el segundo.

Resumiendo, los casos 2 y 4 son equivalentes y los de menos trasiego, seguidos de los casos 1 y 3, en ese orden.

Pregunta 8 (*1 punto*)
Diseñe unos formatos de mensajes aptos para un protocolo de multienvío atómico basado en disentimientos en un medio no fiable, que puede corromper, perder y desordenar mensajes. Éstos pueden ser de tamaño arbitrario. El direccionamiento es punto a punto. No es necesario especificar los tamaños de los campos. Diga para qué se usa cada tipo de mensaje y cada campo.

Hay mensajes de datos, de disentimiento y de asentimiento. Todos pueden llevar a caballo (piggyback) listas de últimos de mensajes entregados de cada origen, para que cuando un interlocutor sepa que un mensaje se ha entregado a todos, borrarlo de su cola.

- *Los de datos pueden venir de su originador o de otro al que se le han solicitado. Tienen:*

    1. *Tipo de mensaje (para distinguir).*
    2. *Dirección de destino (para encaminar).*
    3. *Dirección de origen.*
    4. *Dirección original del mensaje.*
    5. *Número de secuencia del mensage en su origen.*
    6. *Longitud del mensaje (para delimitarlo).*
    7. *Mensaje.*
    8. *Tantos pares (origen, secuencia) como interlocutores, indicando el último mensaje entregado de cada interlocutor.*
    9. *Redundancia (para detectar mensajes corruptos).*

- *Los de disentimiento se envían cuando falta un mensaje de determinado origen. Se envía primero al origen verdadero y, si no responde, a los demás. Tienen:*

    1. *Tipo de mensaje (para distinguir).*
    2. *Dirección de destino (para encaminar).*
    3. *Dirección de origen.*
    4. *Dirección original del mensaje que falta.*
    5. *Número de secuencia del mensaje que falta (en su origen).*
    6. *Tantos pares (origen, secuencia) como interlocutores, indicando el último mensaje entregado de cada interlocutor.*
    7. *Redundancia (para detectar mensajes corruptos).*

- *Los de asentimiento sólo necesitan ocasionalmente, para limpiar mensajes almacenados:*

    1. *Tipo de mensaje (para distinguir).*
    2. *Dirección de destino (para encaminar).*
    3. *Dirección de origen.*
    4. *Tantos pares (origen, secuencia) como interlocutores, indicando el último mensaje entregado de cada interlocutor.*
    5. *Redundancia (para detectar mensajes corruptos).*