dit
UPM

EXAMEN FINAL EXTRAORDINARIO DE SISTEMAS OPERATIVOS
(DISTRIBUIDOS)

7 de Septiembre de 2000

PRIMERA PARTE: sin libros ni apuntes, DURACIÓN: 75 minutos

Conteste en el espacio reservado para ello

Apellidos:	Nombre:

Pregunta 1 (*1 punto*)

El núcleo de un sistema operativo convencional incluye un conjunto de funciones para implementar el denominado *sistema de ficheros*. Razonad sobre las ventajas e inconvenientes que implica construirlo fuera del núcleo en un sistema operativo distribuido basado en un modelo servidor/estaciones de trabajo.

Se aligera considerablemente el tamaño y complejidad del núcleo, lo cual es crucial. Adicionalmente, al estar construido como un proceso servidor independiente, se facilita su ejecución en paralelo con otras partes del sistema operativo, ya sea con hardware centralizado multiprocesador, o distribuido, pero en todo caso con más de un procesador. Como desventaja principal, la ejecución del sistema será más lenta que en un sistema monolítico tradicional, debido a los cambios de contexto de proceso requeridos y, sobre todo, al envío de mensajes por la red si el hardware es distribuido.

Pregunta 2 (*0.5 puntos*)

Explique por qué se dice que NFS es difícil de administrar (piense en traslado de directorios de un servidor a otro).

Porque trasladar datos implica cambiar las tablas de montaje en todos los clientes.

Pregunta 3 (*1 punto*)
Explicar cómo el mecanismo de *capabilities* permite mayor flexibilidad en el acceso a recursos como ficheros (paso de las mismas de unos clientes a otros). En particular, ¿qué ventaja principal se obtiene de esta flexibilidad?. También, ¿cómo se ajustan los derechos de acceso de una *capability* a los que realmente se quiere traspasar?

Las *capabilities* permiten que unos procesos transfieran a otros sus derechos de acceso a ficheros pasando éstas y, por lo tanto, ahorrando así el traslado de los datos en sí, o la manipulación de listas de acceso voluminosas. Para ajustar los derechos transmitidos a los que realmente se quiere transferir, el servicio de ficheros ha de exportar una rutina que permita recortar los derechos de acceso de una *capability* determinada, al subconjunto de los mismos que se desea realmente transferir.

Pregunta 4 (*1 punto*)
En los sistemas de ficheros distribuidos, puede resultar ventajoso tener *caches* en los computadores donde residen los procesos cliente. ¿Por qué?. En todo caso, este esquema no está exento de problemas. ¿Cuáles son estos y cómo se intenta minimizarlos?

El problema se produce cuando hay actualizaciones en una cache cliente. Es necesario en ese caso que la actualización se haga *write-through* a la cache del servidor (y posiblemente al disco). Pero, además, hace falta que la actualización se transmita a las otras caches cliente donde pueda estar ese dato. Y ello supone una inversión del flujo de control usual cliente/servidor, al ser el servidor el que llama al cliente. Y si no se hace con cuidado, pueden presentarse problemas de interbloqueo por cadena circular de llamadas.

Pregunta 5 (*0.5 puntos*)
¿Por qué AFS y CODA necesitan interceptar muchas menos llamadas al sistema que NFS?

Porque las operaciones ordinarias de lectura y escritura y otras se hacen sobre la cache, que es local. Sólo se interceptan apertura y cierre, que son las que necesitan contacto con los servidores.

Pregunta 6 (*0.5 puntos*)
¿Por qué se dice que CODA soporta *replicación optimista*?

Porque no se preocupa de asegurar la coherencia, lo cual es costoso y puede impedir el avance. Le basta asegurarse que se detectarán las incoherencias, que deberían ser pocas.

Pregunta 7 (*0.5 puntos*)
Diga si es *FIFO* el algoritmo de exclusión mutua en anillos explicado en clase. Explíquelo

No lo es, porque obtiene el recurso el que está más cerca del testigo en su dirección de marcha, no el primero que pide el recurso.

Pregunta 8 (*0.5 puntos*)
Diga por qué generalmente es mejor preguntarle un servidor la hora periódicamente a que éste difunda la suya periódicamente.

Preguntando la hora se puede estimar el error máximo, calculando el retardo de ida y vuelta.

Pregunta 9 (*0.5 puntos*)
En sistemas con memoria compartida distribuida basada en la circuitería tradicional de memoria virtual paginada, ¿es preferible invalidar al escribir o actualizar al escribir? Justifíquelo.

Generalmente es preferible invalidar al escribir, ya que actualizar requiere dos faltas de página por escritura (la primera para escribir localmente, ya que la página estará protegida contra escritura, y otra para difundir la modificación, que tendrá lugar al terminar la instrucción, habiendo pusto el procesador en modo traza).

Apellidos:	Nombre:

Pregunta 10
Un proceso sensor lee la temperatura y la guarda en una variable t situada en memoria compartida distribuida. Periódicamente un proceso monitor lee t.

Subpregunta 10.1 (*1 punto*)
Si se usa consistenca de liberación, explique la necesidad de operaciones de sincronización para propagar las actualizaciones de t, a pesar de no ser necesarias a nivel de aplicacion (en un multiprocesador). ¿Cuál de los dos procesos necesita realizar las operaciones de sincronización?

Si no se ejecuta una operación de salida de región crítica, las actualizaciones de t no se propagan. Por tanto el proceso sensor debe ocupar una región crítica antes de escribir en t y liberarla después. El monitor no necesita realizar ninguna operación de sincronización.

Subpregunta 10.2 (*1 punto*)
¿Y si se usa consistencia de entrada?

Si no se ejecuta una operación de entrada en región crítica, las actualizaciones de t no se propagan. Por tanto el proceso monitor debe ocupar una región crítica antes de leer t y liberarla después. El sensor no necesita realizar ninguna operación de sincronización.

Pregunta 11 (*2 puntos*)
Con la ayuda de un ejemplo sencillo y concreto de computación distribuida, arguméntese la ventaje principal que presentan los relojes vectoriales sobre los relojes lógicos de Lamport, a la hora de implementar un observador pasivo. Asimismo, evalúese cuál es el coste adicional de implementación asociado.

A diferencia de los relojes lógicos, los relojes vectoriales permiten caracterizar completamente la computación, al implementar una relación de orden parcial que es isomorfa con la relación de orden causal. Ahora bien, desde un punto de vista práctico, lo más importante es que los relojes vectoriales permiten implementaciones que ocupan menos espacio en búferes y tienen menor latencia. Veámoslo con un ejemplo concreto. Sea simplemente un proceso P1 que envía un mensaje a otro proceso P2, y fijémonos, únicamente, en los dos sucesos de envío y recepción de ese mensaje, tal como los observa un observador global pasivo.
Con relojes lógicos, el observador podrá entregar inmediatamente el mensaje de notificación que viene del suceso envío en el proceso P1, al traer un valor unidad para el reloj lógico. Sin embargo, el observador no podrá entregar el mensaje de notificación del suceso de recepción en el proceso P2, al traer éste un valor de reloj lógico 2 y no haberse recibido por el observador ninguna notificación de suceso, en P2, con valor de reloj lógico uno. Por ello, la entrega de este mensaje de notificación habrá de esperar a que se den las condiciones necesarias para ello (que dependerán de cuales sean los supuestos adicionales que se tengan para la computación distribuida: un plazo máximo de tiempo de transmisión de mensajes, por ejemplo), consumiéndose espacio de almacenamiento e incrementándose la latencia de la comunicación con el observador.
Por contra, ese mismo suceso de recepción en P2, con relojes vectoriales tendrá la hora vectorial (1,1), lo cual permitirá al observador pasivo entregar su mensaje de notificación justo después de haber entregado el del suceso de envío del mismo mensaje, sin mayor dilación ni ocupación de espacio de memoria.
El coste adicional de implementar relojes vectoriales, está claro, es el mantenimiento de los relojes vectoriales en sí en cada uno de los procesos, así como su transmisión por la red, pues en las aplicaciones prácticas el número de procesos puede ser muy alto.