dit
UPM

SOLUCIÓN AL EXAMEN DE SISTEMAS OPERATIVOS (DISTRIBUIDOS)

30 de Enero de 2002

PRIMERA PARTE: sin libros ni apuntes, DURACIÓN: 60 minutos

Respete el espacio especificado para cada respuesta.

Pregunta 1 (*1 punto*)

Explique brevemente un algoritmo de exclusión mutua que use relojes de Lamport, explicando claramente su función.

En el algoritmo de Ricart Agrawala cada proceso demandante pide el recurso a los demás (y a sí mismo) y espera contestaciones de todos, sólo cuando éstas lleguen, entrará. Cuando un proceso recibe una petición de recurso, lo concederá si no lo ha pedido o, si lo ha hecho, no lo pidió el primero. Los relojes de Lamport se usan para que todos los procesos tengan una idea global del orden de las peticiones, por lo que a éstas se les adjunta el reloj de Lamport, resolviendo empates con número de procesador, ya que el orden ha de ser total.

Pregunta 2 (*1 punto*)

¿Qué es una vista en un sistema de comunicación en grupo? ¿Cuál es el requisito básico en el orden de entrega de las vistas?

Una vista es una lista ordenada que contiene los procesos que pertencen a un grupo.

Las vistas se entregan con order total. Si un proceso p *entrega una vista* $v(g)$ *y posteriormente entrega la vista* $v'(g)$*, entonces ningún proceso* $q \neq p$ *entregará* $v'(g)$ *antes que* $v(g)$*.*

Pregunta 3 (*1 punto*)

Justificar la aptitud de los modelos de servicios tolerantes a fallos activo y pasivo para tolerar fallos bizantinos.

El modelo de replicación pasiva se basa en la existencia de un gestor de réplicas primario y un conjunto de gestores secundarios. El primario recibe y ejecuta los servicios solicitados por los frontales. Posteriormente distribuye la informacion de estado a los secundarios.

Este modelo no tolera fallos bizantino. Si falla el primario, no hay forma de detectar este tipo de fallo.

En el modelo de replicación activa, los gestores de réplicas desempeñan papeles equivalentes. Los frontales multienvían la petición y cada réplica la procesa de forma independiente y responden.

Este modelo tolera fallos bizantinos porque el frontal puede recopilar y comparar las respuestas que recibe.

Pregunta 4 (*1/2 punto*)

¿Se puede alcanzar acuerdo entre dos nodos si la red puede fallar perdiendo mensajes? ¿Por qué?

No se puede, no, y lo argumentaremos por contradicción. Pues supongamos que entre dos procesos existe una secuencia finita de mensajes —lo único que los procesos de un sistema distribuido pueden intercambiarse— con la que pueden alcanzar acuerdo, y fijémonos en el último mensaje de esa secuencia, llamémosle m*. Está claro —éste es el punto crucial— que puesto que el proceso que envió* m *no va a poder saber si llegó a su destino o no —al ser el último de la secuencia no hay posibilidad de que su recepción pudiera ser confirmada mediante otro mensaje posterior—, ese último mensaje* m *no ha podido utilizarse para alcanzar el acuerdo, y por lo tanto éste ha tenido que alcanzarse con una secuencia que es la anterior, menos su último mensaje.*

Y repitiendo el mismo razonamiento anterior tantas veces como sea necesario —ésta es la fase "inductiva" de la demostración, y tiene fin porque la secuencia de mensajes es finita, éste es otro punto crucial— se llega a la conclusión de que en realidad el acuerdo se ha alcanzado sin intercambiar ningún mensaje, lo cual es a todas luces imposible y permite concluir que se ha encontrado, por ello, la contradición buscada.

Pregunta 5 (*1 punto*)

Describir una forma en la que sobre un S.O se puede emular otro.

*Para emular un sistema operativo sobre otro se puede utilizar, por ejemplo, la técnica del "trampolín": las llamadas del sistema emulado van directamente al núcleo emulador, y éste, que es capaz de distinguirlas de sus propias llamadas, las reencamina como "llamadas ascendentes" (*upcalls*) a las rutinas correspondientes, que residen en el nivel de usuario e implementan las llamadas del sistema emulado —apoyándose éstas cuando resulta necesario, eso sí, en llamadas propias del sistema emulador—.*

dit
UPM

SOLUCIÓN AL EXAMEN DE
SISTEMAS OPERATIVOS (DISTRIBUIDOS)
30 de Enero de 2002
SEGUNDA PARTE: pueden utilizarse libros y apuntes, DURACIÓN: 70 minutos
Respete el espacio especificado para cada respuesta.

Pregunta 6

En cierto sistema de memoria global compartida con cuatro computadores se implementa un compresor de datos óptimo que lee los datos sin comprimir a una variable compartida $d[0]$. Luego lanza en paralelo tres algoritmos de compresión distintos en tres maquinas distintas, que toman los datos originales y los devuelven comprimidos, cada uno según su algoritmo, en un una variable distinta: $d[1], d[2], d[3]$. El proceso principal espera a que todas las compresiones terminen y luego escribe el resultado del algoritmo que haya dado lugar a mayor compresión. En un C paralelo, el código podrá tener este aspecto:

```
typedef struct { /* Este tipo representa un dato o un resultado */
    unsigned int longitud;      /* Longitudes útiles */
    char datos[1023996];        /* Sitio a ocupar */
} buf;

buf d[4];                       /* Sitio para los datos y los resultados */

/* ... */

lee(&d[0]);     /* Lee los datos a comprimir */
paralelo {      /* Ejectuta las sentencias sigientes en paralelo, en otros procesadores */
    algoritmo1(&d[0], &d[1]);
    algoritmo2(&d[0], &d[2]);
    algoritmo3(&d[0], &d[3]);
} /* Espera a que las sentencias paralelas terminen */
escribe(&d[minimo(&d)]); /* Escribe los datos con mínma longitud */

/* Nota, en C, la experesión &v significa pasar la dirección de v */
```

Subpregunta 6.1 (*1 punto*)

Suponiendo que a todas las variables y rutinas (incluido el programa principal) se les asigna memoria alineada con comienzo de página, que las páginas son de 1024 octetos (`char`) y que un entero sin signo (`unsigned int`) ocupa 4 octetos. ¿Cómo quedará la memoria de cada uno de los procesadores al final de la ejecución si el mejor algoritmo fué el tercero, se usa invalidación al escribir y no hay necesidad de reemplazar páginas por escasez de memoria? Haga explícitas las suposiciones que estime oportunas.

1. *Programa* `main`, *rutinas* `lee`, `escribe` *y* `minimo`, *todo* `d[0]`, *todo* `d[3]` *y la primera página de* `d[1]` *y de* `d[2]`.
2. *Rutina* `algoritmo1`, *todo* `d[0]` *y todo* `d[1]`.
3. *Rutina* `algoritmo2`, *todo* `d[0]`, *todo* `d[2]`.
4. *Rutina* `algoritmo3`, *todo* `d[0]` *y todo* `d[3]`.

Nota: cuando se dice todo d[i], *queremos decir* todas las páginas útiles de d[i], *es decir, aquéllas en las que se ha escrito algo (aproximadamene* `longitud/1024+1`*).*

Subpregunta 6.2 (*0.5 puntos*)

¿En qué variaría el resultado si se usa actualización al escribir?

El resultado es el mismo.

Si se considerara que el programa principal es propietario de todas las páginas (lo que no es muy razonable) el procesador 0 podría tener copia de todos los `d[i]`*.*

Subpregunta 6.3 (*0.5 puntos*)

¿Cuál sería más eficiente: actualización al escribir o invalidación al escribir? Justifíquelo.

Daría igual, ya que mientras se escribe en `d[1]`*,* `d[2]` *o* `d[2]`*, los procesos que lo hacen son los únicos propietarios.*

Si se considerara al programa principal propietario de todos los `d[i]`*, la actualización al escribir sería muy ineficiente, ya que habría que ir actualizando su copia mientras se comprime. Además, en sí misma, la actualización al escribir es muy ineficiente en un sistema basado en páginas.*

Subpregunta 6.4 (*1 punto*)

¿Hay compartición falsa? Justifíquelo y, si la hubiera, diga como evitarla.

No la hay porque todos los procedimientos empiezan en página nueva y, debido a que cada `d[i]` *empieza en una página distinta.*

Pregunta 7

Formalmente, se dice que un corte C es coherente si $((e \in C) \wedge (e' \rightarrow e)) \Rightarrow (e' \in C)$. Sin embargo, en la definición (gráfica) informal se dice que un corte es coherente si no contiene a la punta de una flecha (mensaje) sin contener también a su base. Se pide:

Subpregunta 7.1 (*1/2 punto*)

Formalizar la definición informal, es decir, expresarla con fórmulas al estilo de la anterior.

Para todo mensaje m, $(receive(m) \in C) \Rightarrow (send(m) \in C)$

Subpregunta 7.2 (*1 punto*)

Demostrar (semi)formalmente que ambas definiciones son equivalentes.

Demostraremos la implicación en cada uno de los dos sentidos:

1. *Supongamos que se cumple que* $((e \in C) \wedge (e' \rightarrow e)) \Rightarrow (e' \in C)$ *y que* $(receive(m) \in C)$. *Puesto que* $send(m) \rightarrow (receive(m)$, *se desprende que* $send(m) \in C$.
2. *Supongamos que se cumple que* $(receive(m) \in C) \Rightarrow (send(m) \in C)$, $(e \in C)$ *y* $(e' \rightarrow e)$. *Si* e' *pertenece al mismo proceso que* e, *entonces* $e' \in C$, *por la forma como se construye un corte* C.

   Y si e' *no pertenece al mismo proceso que* e, *entonces* $(e' \rightarrow e)$ *implica que existe un mensaje* m *tal que* $send(m)$ *es bien* e' *o un suceso* s *que le sucede en su mismo proceso, y* $receive(m)$ *es bien* e *o un suceso* p *que le precede en su mismo proceso. Es decir,* $receive(m) \in C$ *por construcción de un corte,* $send(m) \in C$ *por la regla del subproblema 1x y, de nuevo por construcción de un corte,* $e' \in C$.

Pregunta 8 (*1 punto*)

¿Tiene sentido hablar de observaciones causales en un sistema distribuido con mensajería (uno a uno) no causal? ¿Por qué?

Tiene sentido, sí, pues ambas cosas son independientes ("ortogonales", podríamos decir). Puede por ejemplo existir un sistema en el que se mantienen solo relojes vectoriales —que tal como sabemos son insuficientes para garantizar mensajería uno a uno causal— y en el que estos relojes vectoriales se utilicen por un observador pasivo para realizar una observación causal del sistema (y detectar así, en especial, que la mensajería dentro del mismo no es causal).