SOLUCIÓN DEL EXAMEN FINAL SISTEMAS OPERATIVOS

Grado en Ingeniería del Software, 27 de junio de 2012

Ejercicio 1 (2 pts):

a) (1.5pt) Describa qué son y cómo se implementan los enlaces simbólicos y físicos en un sistema de ficheros como el usado por Linux/UNIX (basada en el uso de nodos-i y tablas de punteros a bloques). Suponga un fichero `/home/user/file.dat` e ilustre el resultado de crear los dos tipos de enlaces sobre él y qué ocurriría con ellos al borrar el fichero original.

Enlace físico: cada uno de los nombres con los que se designa un fichero, tal que permite establecer una relación directa entre el nombre y los atributos del fichero. En sistemas de ficheros UNIX se implementa incluyendo en la entrada de directorio la correspondencia entre el nombre del enlace y el número de i-nodo representativo del fichero; entre los atributos del fichero se incluye un contador de enlaces

Enlace simbólico: cada uno de los nombres que se asocian como "alias" al nombre directo o "enlace físico" de un fichero. Se implementa creando un nuevo tipo de fichero cuyo contenido es el nombre de ruta del fichero al que se asocia.

b) (0.5pt) ¿Qué problemas encontramos a la hora de implementar cada uno de dichos tipos de enlace en el sistema de ficheros FAT?

Problema de enlace físico en FAT: el nombre del fichero es uno de los campos del conjunto de atributos de un fichero, contenidos todos en la entrada de directorio. Disponer de más de un enlace físico supondría disponer de más de una entrada de directorio con los mismos atributos exactamente y mantener la coherencia de tales entradas sería costosísimo.

Problema de enlace simbólico en FAT: pasaría por crear un nuevo tipo de fichero cuyo contenido fuera tratado como una referencia indirecta al nombre de ruta de otro fichero. Es factible y de hecho los fichero de enlace (*.lnk) de Windows tienen un comportamiento semejante.

Ejercicio 2 (3 pts): Resolver el siguiente problema de concurrencia que podemos formular brevemente como "un productor de M-en-M y varios consumidores de 1-en-1".

La historia podría contarse de este modo. Estamos en una pizzería de servicio a domicilio atendida por un cocinero (productor) que normalmente está dentro ocupado cocinando, y que cuando le avisen colocará M pizzas en una mesa para que se sirvan los repartidores. Los repartidores (consumidores) llegan a la pizzería con un pedido pendiente cada uno, y si quedan pizzas en la mesa, tomarán una y se marcharán a entregarla; pero si la mesa está vacía el primero que llegue avisará al cocinero para que reponga M pizzas más y esperará a poder llevarse una.

Las pizzas se modelarán como caracteres; la mesa de intercambio entre cocinero y repartidores se modelará mediante una tubería (pipe) como mecanismo exclusivamente de comunicación, y la sincronización (avisos y esperas) y exclusión mutua se resolverá con semáforos.

a) (1pt) Codificar las creaciones, aperturas e inicializaciones de los recursos de Comunicación y Sincronización en los procesos Cocinero y Repartidor.
b) (1pt) Codificar el comportamiento del Cocinero en un bucle sin fin:
   - Esperar hasta que un Repartidor avise que ya no quedan pizzas en la mesa.
   - Depositar M pizzas en la mesa (actualizar variable global nPizzas y enviar M nuevos caracteres al pipe)
c) (1pt) Codificar el comportamiento de los Repartidores en un bucle sin fin:
   - Si aun hay pizzas en la mesa (nPizzas>0), tomar una (un carácter del pipe) y seguir
   - Si no quedan pizzas en la mesa, avisar al Cocinero y esperar a que ponga M más.

```
#define M 10
nPizzas =0;
aviso    =   Semaphore(0)
mutex    =   Semaphore(1)
pipe(fd);
close(fd[0]) //en el Cocinero
close(fd[1]) //en los Repartidores
```

```
void Cocinero(void){
  char buf[M];

  while(1){

    CocinarPizzas();
    sem_wait(aviso);

    sem_wait(mutex);
    nPizzas=N
    sem_signal(mutex);

    //Poner pizzas en la mesa
    write(fd, ptr, M*sizeof(char));


  }
}
```

```
void Repartidor(void){
  int buf;

  while(1){
    EsperarEncargo();

    sem_wait(mutex);
    //¿Hay pizzas?
    //Avisar si no las hay
    if (pizzas==0)
      sem_signal(aviso);

    sem_signal(mutex);
    read(fd, &buf, 1);

    //Coger una pizza
    sem_wait(mutex);
    pizzas--;
    sem_signal(mutex);

    RepartirPizza();
  }
}
```

Ejercicio 3 (2 pts):

a) (0.5pt) ¿Qué es el TLB (Translation Look-aside Buffer) y para qué se utiliza?

Memoria asociativa con información sobre últimas páginas accedidas. Se usa como cache de entradas de la Tabla de Páginas.

b) (0.5pt) ¿En qué consiste un fallo de página? y ¿por qué el número de fallos de páginas repercute en el rendimiento de un programa?

La MMU produce un fallo de página (trap) cuando una dirección virtual no tiene correspondencia en la Memoria Principal. El SO tiene que traer de Memoria Secundaria (HDD) la información para poder ser accedida. Este proceso muy costoso entiempo si lo comparamos con el acceso a MP.

c) (0.5pt) ¿Cuales son las ventajas e inconvenientes de la E/S programada frente a la realizada a través de interrupciones desde el punto de vista de eficiencia y sencillez? Razone su respuesta.

Programada frente a Interrupciones:

- Ventaja: no hay gasto de tiempo de gestión de interrupción, el tratamiento es más sencillo.
- Desventaja: Consume mucha CPU para dispositivos poco usados, ya que el procesador es el responsable de extraer o enviar datos entre la memoria y el controlador de dispositivo. La interrupción salta cuando los datos están listos y no hay gasto inútil de CPU (más allá del cambio de contexto).

d) (0.5pt) ¿En qué consiste el Acceso Directo a Memoria (DMA)? ¿Por qué se utiliza?

Necesita un controlador con DMA y se usa para evitar la E/S programada de grandes bloques de datos. Evita el uso de la CPU transfiriendo los datos directamente entre los dispositivos de E/S y la memoria

Ejercicio 4 (3 pts): Supóngase un Sistema Operativo que usa como algoritmo de planificación Round Robin con ***q***=5t y que el cambio de contexto ente procesos tarda 1t. Decidimos ejecutar dos procesos multihilo usando una biblioteca de hilos con un planificador que reparte el ***q*** del proceso entre los hilos aplicando también RR con ***q***=1t y sin coste de cambio de contexto entre sus hilos. Los dos procesos tienen el siguiente comportamiento:

Proceso A	Perfil	Proceso B	Perfil
Th1	1CPU; 6E/S; 1CPU	**Th1**	2CPU; 3E/S; 2CPU
Th2	2CPU; 3E/S	**Th2**	3CPU; 5E/S
Th3	2CPU; 2E/S; 6CPU		

a) (2pt) Rellene la siguiente tabla sombreando la casilla correspondiente cuando se esté usando la CPU y ponga una X cuando se esté haciendo Entrada/Salida:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
A-Th1	1	X	X	X	X	X	X				1’																			
A-Th2									1			2	X	X	X															
A-Th3										1								2	X	X		1’	2’	3’	4’	5’				6’
B-Th1			1		2	X	X	X							1’													2’		
B-Th2				1										2		3	X	X	X	X	X									
SO		U						U					U				U				U						U		U	
Idle						U	U												U	U										
T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

a) (0.5pt) Indique el porcentaje de tiempo que la CPU está siendo utilizada, y qué parte del uso es debido a sobrecarga del SO

Uso de CPU:

- 19t de usuario
- 6t de SO
- 4t Idle
- Total: 29t

Uso de CPU (19+7)/30 (86.7%) del cual 7/19 (36.8%) es de SO

b) (0.5pt) ¿Sería diferente la planificación final si los hilos fuesen gestionados por el kernel? ¿Por qué?

Sí, el kernel sería consciente de que son los Threads los que hacen E/S y no quitaría la CPU al proceso, se lo daría a otro Thread del mismo proceso (en caso de poder): Permiten paralelismo con Threads de un mismo proceso bloqueados y en ejecución.