Ingeniería Superior de Informática. Curso 3º.
Sistemas Operativos. Examen Final. TEORÍA.
31 de Enero de 2005

Nombre: _____
DNI: _____

1. **(2 puntos).** La TFA es la tabla que, en UNIX, guarda los punteros de posición de cada fichero abierto. Estos punteros indican el desplazamiento en bytes a partir de cual tiene efecto la próxima operación de lectura (read) o escritura (write) de un proceso sobre el fichero en cuestión designado mediante un descriptor. ¿Qué problemas o limitaciones se plantearían si dichos punteros se almacenasen en:
   a) la tabla de i-nodos en memoria (TIN)?
   b) la tabla de descriptores de ficheros (TDF o Tdaa)?

2. **(2 puntos)**. Considerar un sistema de E/S donde se utiliza una cache de disco con política de sustitución LRU. La cache dispone de M *buffers* y emplea N clases *hash* (el cálculo de clave *hash* se efectúa con la operación de módulo).
   a) ¿Qué utilidad desempeñan las N clases *hash*?
   b) ¿Deben guardar alguna relación el número M de *buffers* y el número N de clases *hash*? Razonar la respuesta.

3. **(2 puntos)**.
   a) ¿Qué recursos comparten los hilos de un mismo proceso?
   b) JAVA es un lenguaje que permite hilos (*thread*) como objeto de programación. ¿Es cierto o falso que la ejecución de un programa JAVA con hilos requiere que el Sistema Operativo del sistema donde se ejecuta admita asimismo la gestión de hilos? ¿Por qué?

4. **(2 puntos)**.
   a) Explica el comportamiento de la instrucción TS.
   b) ¿Qué problemas presenta el uso de TS como medio básico de solución del problema de Exclusión Mutua?

5. **(2 puntos)**.
   a) Se dice que la política de sustitución "Reloj" es una aproximación a la política LRU. Justifícalo.
   b) ¿Es compatible la política de sustitución "Reloj" con el empleo de "buffering de páginas" para gestión de la memoria virtual? ¿Por qué?

Soluciones:

1.
a) Si el puntero se guarda en la TIN todos los procesos que se refieran a un mismo fichero, sea con el descriptor que sea, y con la apertura que sea, manejarían el mismo puntero y cada operación afectaría a la siguiente, provocando el caos en los accesos.
b) Si el puntero se guarda en la TDF, cada descriptor distinto utilizaría un puntero particular y no habría posibilidad de compartir accesos entre dos descriptores de un mismo proceso o entre dos descriptores de procesos distintos.

2.
a) Permiten simular la "asociatividad" de la cache, esto es, encontrar rápidamente si la entrada (bloque) correspondiente a una clave (número de bloque) se halla o no almacenada en alguno de los buffers de la cache. Esto lo logra "asociando" de principio un número de bloque a un número de buffer mediante la operación (NºBufferInicial = NºBloque % NºClasesHash) con la intención de de ligar cada bloque a un buffer distinto si es posible.
b) NºClasesHash >= NºTotalBuffers para tratar de que el tamaño de las colas hash sea, la mayoría de las veces, 1 o 0 con objeto de contestar con una sola consulta a la pregunta ¿está el bloque NºBloque en cache, sí o no?

3.
a) El mapa de memoria, la tabla de descriptores de ficheros, la lista de recursos (semáforos, colas de mensajes, …) , la lista de señales.
b) Falso. Los hilos en Java son hilos de usuario (ULT) soportados por el lenguaje y gestionados por el motor de ejecución (intérprete) de Java. Si el SO subyacente gestiona hilos kernel (KLT), el motor Java normalmente utilizará una estrategia combinada de ULT ejecutados sobre KLTs; pero si el SO no gestiona hilos, el propio JAVA se encargará de gestionarlos de cara al usuario.

4.

```
a) boolean TS(int memoria)  {
        if (memoria == 0) {
                memoria = 1; return TRUE;
        } else
                return FALSE;
}
```

```
P1:     do {esperar } while (! TS(s));
           SECCION CRITICA;
        s = 0;

P2:     do {esperar } while (! TS(s));
           SECCION CRITICA;
        s = 0;
```

b) Espera activa (consumo innecesario de CPU), falta de ordenación de las esperas (FIFO o por prioridades) – política aleatoria de señalización, posibles inaniciones, posible interbloqueo en planificaciones sin expropiación

5.
a) Establece una ordenación parcial entre todas las páginas consideradas en función de si su uso ha sido más o menos reciente. La ordenación distingue dos clases: las páginas con R=0 o LRUs, y las páginas con R=1 o MRUs. Se elige para sustituir una de las páginas LRU, aplicando un sub-criterio FIFO
b) No. "Buffering" determina CUÁNDO se eligen las páginas candidatas a ser sustituidas (cuando el número de tales páginas es inferior a un cierto valor tope), pero no determina CON QUÉ CRITERIO se seleccionan tales páginas. Luego "buffering" es compatible con cualquier política de sustitución.

Ingeniería Superior en Informática. Curso 3º.
Sistemas Operativos. Examen Final. PROBLEMAS.
31 de Enero de 2005

Nombre: _____
DNI: _____

1) **(3 puntos).** Determinar el rendimiento (%uso de CPU y %sobrecarga del S.O.) obtenido al planificar dos procesos multi-hilo en un sistema donde los hilos se gestionan en espacio de usuario. El algoritmo de planificación del S.O. es RR con q=100, y el cambio de contexto ente procesos tarda 20. El planificador de la biblioteca de hilos reparte el q del proceso entre los hilos aplicando RR con q=10 y sin coste de cambio de contexto entre sus hilos. Los dos procesos tienen el siguiente comportamiento:

Proceso A	Proceso B
Hilo 1 (30-CPU; 110-E/S; 40-CPU)	Hilo 1 (20-CPU; 50-E/S; 60-CPU)
Hilo 2 (50-CPU)	Hilo 2 (40-CPU; 110-E/S; 20-CPU)
Hilo 3 (30-CPU)	
Hilo 4 (20-CPU; 60-E/S; 40-CPU)	

2) **(4 puntos)**. Completar el diseño de un monitor que simula el comportamiento de un semáforo con disciplina FIFO, teniendo presente que las colas "*condition*" del monitor actúan con disciplina aleatoria; es decir, codificar el procedimiento **`wait_fifo()`**.

```
monitor semfifo {
      int cnt, ix, jx, r;
      condition s[NP];

      int InitSem(int val) {
            if (cnt<0 || val<0) return ERROR;   // PREGUNTA c)
            cnt = val; ix = 0; jx = 0; return OK;
      }

      int wait_fifo() {
            ...
            ...
      }

      int signal_fifo() {
            cnt = cnt+1;
            if (cnt <= 0) {
                  r = jx; jx = (jx+1)%NP; signalc(s[r]);
            }
            return OK;
      }
}
```

a) ¿Qué representan las variables del monitor: **`cnt`**, **`ix`**, **`jx`**; y la constante **`NP`**?
b) ¿Qué expresión nos permitiría conocer el número de procesos actualmente en espera en la cola FIFO del semáforo sin necesidad de mantener un contador adicional exprofesso?
c) ¿Qué sentido tiene la sentencia marcada y qué comportamiento indeseable podría tener un programa que invocara a **`InitSem()`** si la eliminásemos?

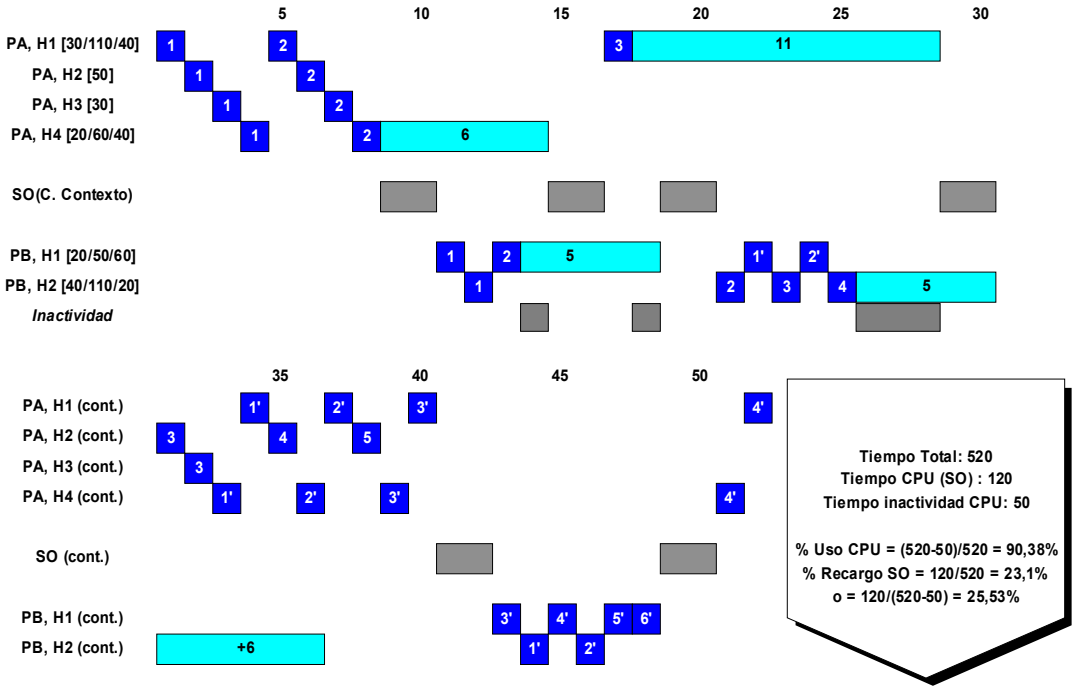
3) **(3 puntos)**. Se dispone de un disco de 1,2 MB de capacidad formateado para que trabaje con un sistema de ficheros tipo UNIX cuyas características se describen a continuación:

- Tamaño de bloque: 512 bytes.
- Tamaño de la dirección de los bloques: 2 bytes.
- Numero de i-nodos : 50
- Campos de un i-nodo:
    - Atributos del fichero:
        - Id. del Propietario y del grupo.
        - Permisos de lectura para dueño, grupo y resto del mundo.
        - Permisos de escritura para dueño, grupo y resto del mundo.
        - Permisos de ejecución para dueño, grupo y resto del mundo.
    - Contador de enlaces (1 byte).
    - 2 punteros directos, 2 indirectos simple y 1 indirecto doble.

Responder **razonadamente** a las siguientes preguntas:

a) ¿Que tamaño máximo podrá tener un fichero almacenado en este disco?
b) Número máximo de ficheros que puede tener este sistema en función de las características expuestas.
c) Suponiendo que el sistema de ficheros admita enlaces duros y simbólicos:
    c1). ¿Qué número de enlaces duros podríamos crear del fichero "`contenido`" que se encuentra en el directorio raíz y es el único fichero del sistema?
    c2). ¿Y de enlaces simbólicos?

Soluciones:
1.



2.

```
int wait_fifo() {
      cnt = cnt-1 ;
      if (cnt < 0) {
            if (-cnt > NP) { cnt = -NP; return ERROR;}
            r = ix; ix = (ix+1)%NP;
            waitc(s[r]);
          }
          return OK;
}
```

a) **cnt**: valor de cuenta del semáforo si es positivo, y número de procesos (valor absoluto) en espera del semáforo si es negativo
ix, jx: índices al array **s** de variables *condition* que actúa como una cola FIFO para el semáforo; **ix** el índice de entrada a la cola FIFO y **jx** el índice de salida de la cola FIFO
NP: el numero máximo de procesos que pueden esperar a la vez por el semáforo; el tamaño de la cola FIFO de espera, por tanto.

b) **n_procesos_esperando = cnt < 0 ? -cnt : 0;**

c) si se reinicializa el semáforo después de haberlo usado algún tiempo, evita variar el valor del semáforo en caso de que haya algún proceso esperando; si no fuese así, los procesos en espera se quedarían esperando por siempre

3.
a) Tamaño máximo de fichero
Por cuenta de punteros a bloque: $2 + (512/2) \times 2 + (512/2)^2 = 2 + 512 + 65536 = 66050$ bloques
Por espacio de disco en bloque: $1{,}2 \times 1024 \times 1024 / 512 = 24576$ bloques
Por direcciones de bloque posibles: $2^{16} = 65536$ bloques
Si no existiese asignación retardada, el máximo vendría impuesto por la capacidad del disco
Con asignación retardada, el máximo lo impone el número de direcciones de bloque posibles

b) Número máximo de ficheros = nº de inodos = 50

c1) Número máximo de enlaces “hard” depende del contador de enlaces = $2^8 - 1 = 255$

c2) Número máximo de enlaces “symbolic” (en este volumen) depende del nº de inodos ya que cada enlace utiliza un inodo. Como hay 50 inodos totales y el directorio raíz ocupa uno y el fichero “contenido” ocupa otro, el número de inodos que queda para enlaces es 50 – 2 = 48