Ingeniería Informática. Curso 3º.
Sistemas Operativos
Examen Final. TEORIA.
9 de Febrero de 2007

Planteamiento del examen de TEORIA:

El examen constará de 5 preguntas/cuestiones que pueden ser libremente escogidas con la restricción de que debe haber una por cada tema del programa de la asignatura. Todas las preguntas puntúan igual.

Indicar claramente, en la primera hoja de examen la numeración de las preguntas escogidas (p.ej., 1, 4, 6, 8 y 9) acompañando al nombre y apellidos, número de DNI y grupo correspondiente del examinando.

1. **[PROCESOS]** ¿Qué esquema jerárquico de procesos genera este programa?

```
main() {
      for (int i=0; i<N; i++)
            if (i%2 == 0) {
                  fork();  wait(NULL);
            }
}
```

¿Cuántos procesos de esta jerarquía estarán presentes simultáneamente en memoria como máximo?

2. **[PROCESOS]** En planificación LINUX un proceso puede estar sometido a una de tres políticas: SCHED_FIFO, SCHED_RR o SCHED_OTHER- ¿Cuál(es) de ellas corresponden a políticas expropiativas y cuál(es) son las causas de expropiación en cada caso?
¿Qué definición de *quanto* es aplicable a cada política?

3. **[SINCRONIZACION Y COMUNICACIÓN]** El mecanismo ***pipe*** ¿es de tipo comunicación, sincronización o mixto? Explicar.
¿Qué ocurre cuando se ordena leer de un ***pipe*** y no hay descriptores de escritura abiertos sobre el mismo?
¿Qué ocurre cuando se ordena escribir de un ***pipe*** y no hay descriptores de lectura abiertos sobre el mismo?

4. **[SINCRONIZACION Y COMUNICACIÓN]** ¿Qué caracteriza al mecanismo de paso de mensajes de tipo cita ("*rendezvous*")? ¿Qué ventajas ofrece? ¿Cómo podrías simular este comportamiento con el mecanismo de paso de mensajes POSIX?

5. **[ENTRADA/SALIDA]** La estrategia N-SCAN de planificación de disco es una variante de SCAN en la cual se dispone de un número indefinido de colas de peticiones de tamaño N cada una de ellas. Las primeras N peticiones van a la cola 0, las N siguientes a la cola 1, etc. Inicialmente se procesan las peticiones de la cola 0 con política SCAN; cuando se vacía, se procede con las peticiones de la cola 1, y así sucesivamente.
Si N = 1, ¿qué tipo de planificación se está aplicando en realidad?
Si N = ∞, ¿qué tipo de planificación se está aplicando en realidad?
¿Qué se logra utilizando un N determinado intermedio, por ejemplo N=10?

6. **[ENTRADA/SALIDA]**

```
fd = open("/dev/tty0", O_RDONLY);
read(fd, buffer, 10);
```

Esta operación de ***read()*** se resuelve por parte del sistema operativo realizando la entrada de 10 caracteres desde el teclado del terminal conectado a una línea serie específica. ¿Cómo sabe el S.O. qué manejador de dispositivo debe emplear y a qué dispositivo concreto va dirigida la llamada? ¿Podrá utilizar la *cache* de *buffers* para evitar el acceso de E/S?

7. **[SISTEMAS DE FICHEROS]** De los 6 tipos básicos de organización de discos RAID, ¿qué tres son los menos adecuados para su uso práctico y por qué?
Comparar en términos de rendimiento estas dos organizaciones: un disco doble en disposición RAID0 con dos particiones cada una de las cuales aloja un sistemas de ficheros diferente frente al uso de dos discos separados cada uno de los cuales soporta un sistema de ficheros distinto. ¿Qué es preferible?

8. **[SISTEMAS DE FICHEROS]** Consideremos un sistema de ficheros *ext2* con la organización: Superbloque (SB), Mapa de bits de inodos (MN), Mapa de bits de bloques (MB), Tabla de inodos (BN), Bloques de Datos (BD).
Indicar los accesos elementales de escritura a disco que debe realizar el *kernel* para ejecutar la llamada al sistema

```
mkdir("/tmp/A");
```

suponiendo que el directorio **`/tmp`** está inicialmente vacío, y razonar la situación correcta o errónea en que quedaría el sistema de ficheros donde se crea el directorio, si el sistema cayera tras cada uno de tales accesos.

9**. [MEMORIA]** Dado el siguiente trozo de programa en ensamblador:

```
1020  LD   R0, 6144
      PUSH R0
      CALL 5120
      SUB  SP, #16
      CMP  0(SP), #4
      JEQ  5152
      -----
5120  INC  R0
      ST   R0, 6148
      RETURN
```

a) Deducir la traza de referencias, la traza de páginas (para un tamaño de página de 1KB) y la traza de regiones que genera.
(Nota.- cada instrucción ocupa una palabra de 4 bytes; SP es el registro puntero de pila y R0 es un registro general, ambos de longitud 4 bytes)
b) Si se empieza sin ninguna página cargada en memoria, ¿cuántos fallos de página se producen?

10**. [MEMORIA]** La hiperpaginación ("*thrashing*") es una situación indeseable a la que puede llevar la gestión de memoria virtual.
¿Qué síntomas de comportamiento del sistema revelan tal situación?
¿Qué medidas serían efectivas para remediarla?

Ingeniería Informátiuca. Curso 3º.
Sistemas Operativos
Examen Final. PROBLEMAS.
9 de Febrero de 2007

1. Hay que resolver un problema de acceso concurrente, por grupos, a una sección crítica donde se comparte el recurso R. Cada hilo de ejecución ***H(i)*** pertenece a un grupo de los N existentes en la aplicación (el argumento ***i*** indica el grupo, desde ***i*** igual a 0 hasta N-1).
   Los hilos pertenecientes a un mismo grupo pueden estar concurrentemente dentro de sus respectivas secciones críticas utilizando el recurso R, pero no podrán compartir el acceso con procesos de un grupo diferente.
   Cuando todos los hilos de un mismo grupo abandonen la sección crítica, entrarán todos los hilos que haya en espera, pertenecientes al grupo siguiente en orden circular. El esquema del programa es el siguiente

```
#define N ...

void proceso(int i_grupo) {
   while (true) {
      Procesamiento local;
EntrarSC(i_grupo);
      Sección Crítica;
      SalirSC(i_grupo);
    }
}
```

Monitor.c

```
... // (A)  Declaraciones

EntrarSC(grupo) { // (B)¿Está libre la S.C.
             // o está ocupada por mi grupo?
       //   SI → pasar; NO → esperar
}
SalirSC(grupo) {  // (C) ¿Soy el último de mi grupo
             // en abandonar la S.C.?
       //   NO → continuar
            //   SI → dejar pasar al siguiente grupo en espera,
            //     si no espera nadie, liberar el recurso R
}
```

Completa el código del monitor {secciones (A), (B), y (C)} para que el programa cumpla las especificaciones de funcionamiento indicadas.
Nota.- Emplear la operación **`empty(vcond)`** que devuelve TRUE si la cola de la variable de condición **`vcond`** está vacía.

Puntuación: Sección A (1 punto); Sección B (2 puntos), Sección 3 (2 puntos)

Recordatorio: un monitor es un fichero de compilación independiente donde se define un conjunto de funciones públicas que se ejecutan en exclusión mutua entre ellas para lograr lo cual se codifican haciendo uso de mutexes y variables condicionales.

2. Considere el código de los siguientes procesos:

```
int fda;
Proceso A() {
  char buf[1024];
        fda = open("tmp/faa");
//código (1.5ms)
1       read(fda,buf,100);
        //código (1.5 ms)
create_thread(,Hilo_Nuevo) 2
read(fda,buf,800);
        //código (1.5 ms)
         lseek(fda,800,SEEK_CUR);
3       read(fda,buf,300);
}

Hilo_Nuevo() {
        //código( 1 ms)
4     read(fda,buf,800);
        //código (2 ms)
5      read(fda,buf,700);
}
```

```
Proceso B() {
        char buf_w[1024]="cccc…";
        char buf[1024];
        int  fdb;

         //código ( 1 ms)
         fdb = open("tmp/fbb");
6        read(fdb,buf,250);
         // código (2.25 ms)
         lseek(fdb,2000,SEEK_CUR);
7        read(fdb,buf,250);
          // código (1ms)
8        write(fdb,buf_w,1000);
      }
```

Se ha elegido **FAT** como sistemas de ficheros de la partición de trabajo. El tamaño de bloque (físico y lógico) de 1KB (1024 bytes). A continuación se muestra la información relevante del estado inicial del sistema de ficheros:

Estado inicial de la FAT: 7 2 6 0 3 free eof eof .....

Contenido del fichero de directorio **tmp**:

Nombre	Primer_bloque	Tamaño	……..
faa	4	3920	
fbb	1	2870	

El planificador del SO utiliza una política **Round Robin** con un ***quanto* de tiempo de 1ms**.
Las llamadas al sistema read/write son **bloqueantes** y **tardan 1ms por cada bloque de físico** accedido.
Cuando un proceso finaliza una operación de entrada/salida, pasa al final de la cola de *Listos*. El proceso A comienza su ejecución un instante antes que el proceso B.

La gestión de hilos se realiza a nivel de *kernel*. Cuando se crea un hilo, el nuevo hilo se coloca al final de la cola de *Listos*, y el hilo *creador* continúa con su ejecución. Considere que las llamadas *open* ,*lseek,*y *create_thread* tardan **0ms** en ejecutarse.

Se pide:

a) (1,5 punto) Determinar el estado final de la FAT y de la entrada de directorio "**tmp**"
b) (1,5 punto) Realizar un diagrama temporal con la planificación de los hilos/procesos, indicando el estado de cada hilo. Calcular el *tiempo de retorno* y de *espera* de cada hilo/proceso.
c) (1 punto) Para cada llamada a "read" y "write", indicar los bloques lógicos y físicos accedidos.
d) (1 punto)Se dispone de una *cache de buffers* con 4 *buffers* y 2 clases *hash*. Dicha cache, aplica la política de sustitución LRU y emplea como algoritmo de cálculo de clave *hash* la operación ***módulo***. Inicialmente, la cache de bloques está vacía. En caso de acierto en la cache de bloques, una lectura será NO bloqueante y tardará únicamente 0.1ms. Determinar nuevamente la planificación teniendo en cuenta los aciertos/fallos de cache, y mostrar el estado final de la cache de buffers.

SOLUCIONES TEORIA

1. [PROCESOS]
    a. El esquema jerárquico resultante es un árbol con raíz en el proceso original con 2^{N+2} hijos de modo que el número total de procesos contabilizados es $2^{(N/2+1)}$; cada uno de los hijos es raíz de un árbol semejante con 2^i elementos para i=0..N/2
    b. Puesto que cada proceso activo solo puede tener un hijo activo al usar la combinación {fork(); wait(NULL);} simultáneamente solo podrán estar los procesos que formen una ruta en el árbol empezando en la raíz y acabando en cualquiera de los procesos generados. La ruta más larga está formada por N/2 +1 procesos.
2. [PROCESOS]
    a. Todas son políticas expropiativas. SCHED_FIFO por prioridad (la suya es fija); SCHED_RR por prioridad (la suya es fija) y por agotamiento del quanto; SCHED_OTHER por prioridad (la suya es dinámica) y por agotamiento del quanto
    b. SCHED_FIFO no emplea quantos; SCHED_RR usa un quanto fijo; SCHED_OTHER usa un quanto variable que además se interpreta como prioridad
3. [SINCRONIZACION Y COMUNICACIÓN]
    a. Es un mecanismo MIXTO de comunicación y sincronización
    b. El proceso lector recibe un resultado 0 que debe interpretar como Fin_de_Fichero.
    c. El proceso escritor recibe una señal SIGPIPE que si no se captura causa la terminación del proceso.
4. [SINCRONIZACION Y COMUNICACIÓN]
    a. En una "cita" tanto la operación send() como la operación receive() son bloqueantes
    b. Tres ventajas: Sincronización mutua; garantía de entrega y recepción de mensajes; no tiene necesidad de copia intermedia de datos
    c. El mecanismo de mensajes POSIX usa send() no bloqueante y receive() bloqueante. Para simular una "cita" el emisor debería codificar
        send(mensaje_datos); receive(confirmación)
        y el receptor debería codificar
        receive(mensaje_datos); send(confirmación)
5. [ENTRADA/SALIDA]
    a. Con N=1 la estrategia resultante es la FIFO
    b. Con N=infinito la estrategia resultante es la SCAN pura
    c. Con N=10 se logra limitar el ámbito de aplicación de la estrategia SCAN a las primeras 10 peticiones pendientes como máximo y como consecuencia limita el tiempo de espera o relegación que puede afectar a cualquier petición
6. [ENTRADA/SALIDA]
    a. Al consultar en la apertura (open()) los atributos del fichero el S.O. advierte que se trata de un fichero especial de dispositivo (en este caso del tipo "caracteres") que dispone de un número MAYOR y un numero MENOR. El número MAYOR sirve de índice para referenciar el módulo manejador (driver) que determina el comportamiento del dispositivo (en este caso un terminal serie) y el número MENOR se interpreta como la numeración del dispositivo concreto al que debe dirigirse
    b. NO. La cache de buffers es utilizada por dispositivos de bloques y no por dispositivos de caracteres como es este caso (/dev/tty0)
7. [ENTRADA/SALIDA]
    a. RAID-2 (Paridad bit Hamming), RAID-3 (Paridad bit) y RAID-4 (Paridad en un disco dedicado). En los dos primeros casos porque se exige la operación solidaria de todos los discos componentes lo que eleva las exigencias técnicas de operación; en el tercer caso, el disco de paridad dedicado se convierte en un cuello de botella ya que todas las operaciones de escritura exigen su concurso.
    b. Organización RAID0: ambos sistemas de ficheros están distribuidos por ambos discos; por tanto, la carga de trabajo (accesos) está equilibrada entre ambos sea cual sea la conjunción de ficheros a los que se acceda en cada momento
        Organización de discos separados: cada sistema de ficheros está alojado en un disco dedicado. Por tanto, para que exista un buen reparto de cargas entre ambos los ficheros a los que se accede deben estar repartidos entre ambos sistemas de ficheros y esto no se puede garantizar, en general.
        Es preferible, pues, RAID0
8. [ENTRADA/SALIDA]

9. [MEMORIA]
    a. Traza de referencias:
        1020, 6144, 1024, (SP), 1028, 5120, 5124, 5128, 6148, 1032, 1036, (SP-4-16), 1040

        Traza de páginas
        0, 6, 1, x, 1, 5, 5, 5, 6, 1, 1, x ó x-1, 1

        Traza de regiones
        T, D, T, P, T, T, T, T, D, T, T, P, T
    b. 5 ó 6 fallos: páginas 0, 1, 5, 6, x (y quizás x-1)
10. [MEMORIA]
    a. Síntomas: bajo %uso de CPU y elevado %uso (ocupación de ancho de banda) de dispositivo de swap
    b. Remedios: Controlar la carga (retirar procesos de memoria, permitiendo ampliar el espacio residente de los procesos activos), liberar memoria dedicada a otros usos (p.ej. a la cache de buffers) o aumentar la capacidad de memoria (esta es una medida de configuración hardware y no es decisión del S.O.)

SOLUCION PROBLEMAS

1.

A. DECLARACIONES

```
mutext_t monitor;          // mutex para exclusión mutua del monitor completo
vcond_t espera[N];         // una cola de espera para cada grupo diferente
int ndentro = 0;           // contabiliza cuántos procesos están usando el recurso
int grupo_dentro;          // identifica el grupo a que pertenecen los procesos que están
                           // usando actualmente el recurso
```

B. Entrar_SC(grupo)

```
lock(monitor);
if (ndentro == 0 | | grupo_dentro == grupo)
      grupo_dentro = grupo;
else
      wait(espera[grupo], monitor);
ndentro++;
unlock(monitor);
```

C. Salir_SC(grupo);

```
lock(monitor);
ndentro--;
if (ndentro == 0) {
      for (int i=0; i<N; i++)
            if (!empty(espera[(grupo+i+1) % N]) break;
      if (i != N) {
            grupo_dentro = (grupo+i+1)%N;
            broadcast(espera[grupo_dentro]);
      }
}
unlock(monitor);
```

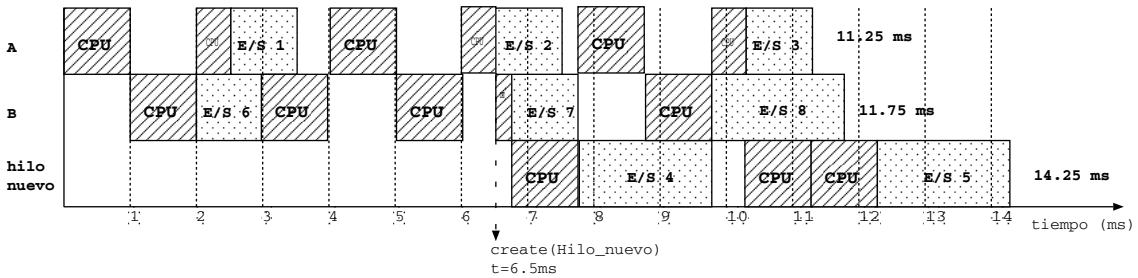
2.

a) FAT Final: 7 2 6 0 3 eof 5 eof

Contenido final del directorio **tmp**:

Nombre	Primer_bloque	Tamaño	……..
faa	4	3920	
fbb	1	3500	

b)



Tiempos de espera:

Proceso A: 3.75ms Proceso B: 3.5ms Hilo Nuevo: 0.75ms

c)

Proc.	Op.	Puntero (inicio – fin)	Bloque(s) lógico(s)	Bloque(s) físico(s)
A	1	0 – 100	0	4
	2	100 – 900	0	4
	4	900 – 1700	0,1	4,3
	3	2500 – 2800	2	0
	5	2800 – 3500	2,3	0,7
B	6	0 – 250	0	1
	7	2250-2500	2	6
	8	2500 – 3500	2,3	6,5

d)

Op.	Bloque(s) físico(s)	Clave Hash (mod2)	Fallo/Acierto	Nº Buffer accedido
6	1	1	F	0
1	4	0	F	1
2	4	0	A	1
7	6	0	F	2
4	4,3	0,1	A,F	1,3
8	6,5	0,1	A, F (1)	2,0
3	0	0	F (4)	1
5	0,7	0,1	A, F(3)	1,3

Contenido final de la cache de buffers: 5 0 6 7 (índices de bloques físicos)
La planificación final apenas se ve alterada.