Universidad Carlos III de Madrid	**Departamento de Informática** **Grado en Ingeniería Informática** **Sistemas Operativos** **Examen de la convocatoria ordinaria** **20 de enero de 2011**	ARCOS

ATENCIÓN:

- Lea atentamente todo el enunciado antes de comenzar a contestar.
- Dispone de **3 horas** para realizar la prueba.
- No se podrán utilizar libros ni apuntes, ni calculadoras de ningún tipo.
- Los teléfonos móviles deberán permanecer desconectados durante la prueba (apagados, no silenciados).
- Solamente se corregirán los ejercicios contestados con bolígrafo. Por favor no utilice lápiz.

APELLIDOS:
NOMBRE:
NIA:
GRUPO:

Ejercicio 1. Teoría [2,5 puntos]:

Pregunta 1.
¿Cuándo entra un proceso en estado zombie?
A.- Cuando muere su padre y él no ha terminado todavía.
B.- Cuando muere su padre sin haber hecho `wait` por él.
C.- Cuando él muere y su padre no ha hecho `wait` por él.
D.- Cuando él muere y su padre no ha terminado todavía.

Razone la repuesta.

Pregunta 2.
¿Cuál de las siguientes políticas de planificación es más adecuada para un sistema de multiproceso de tiempo compartido?
A.- Primero el trabajo más corto.
B.- Round-Robin.
C.- Prioridades.
D.- FIFO.

Razone la repuesta.

Pregunta 3.
¿Cuál es la diferencia entre nombre absoluto y relativo? Indique dos nombres relativos para `/ssoo/alumnos/examen/enunciado`. Indique el directorio respecto al que son relativos.

Ejercicio 2 [2,5 puntos]:

La **criba de Eratostenes** es un algoritmo para generar una serie de números primos consecutivos. Permite hallar todos los números primos menores que un número natural

dado *N*. Para ello se forma una tabla con todos los números naturales comprendidos entre 2 y *N* y se van tachando los números que no son primos de la siguiente manera: cuando se encuentra un número entero que no ha sido tachado, ese número es declarado primo, y se procede a tachar todos sus múltiplos. El proceso termina cuando el cuadrado del mayor número confirmado como primo es mayor que *N*.

En este ejercicio se pide implementar la criba de Eratóstenes con ***procesos y pipes***.
El algoritmo debe funcionar de la forma siguiente:

- Un padre crea N hijos.
- El padre se conecta con el hijo 0.
- El hijo **i** se conecta con los hijos i-1 e i+1.
- El padre genera una serie de números consecutivos 2, 3, 4, 5, … y los envía uno a uno al hijo 0. La secuencia termina con el número n-1.
- Cada hijo almacena el primer número que recibe en una variable primo_local. Posteriormente:
  - si el número que recibe NO es múltiple de primo_local, lo reenvía al siguiente hijo.
  - si el número recibido es múltiplo de primo_local, lo filtra y no hace nada.
- Cuando un hijo recibe un -1, lo pasa al hijo siguiente y termina.
- El padre espera a que terminen todos los hijos antes de terminar.

En la siguiente tabla se muestra un ejemplo de funcionamiento de cada hijo creado:

	Hijo 0	**Hijo 1**	**Hijo 2**	**Hijo 3**	**Hijo 4**	**…**
Primo local =	1	2	3	5	7	
Filtrados:		4,6,8,10…	9,15,21,27…	25,35…	49…	

Ejercicio 3 [2,5 puntos]:

Se tiene un **array de 100** elementos en el que se quiere realizar varias **iteraciones** en cada una de las cuales se debe colocar en **cada casilla la media de la suma del contenido de esa casilla y sus dos adyacentes**.

Es decir que para cualquier casilla del array entre 1 y 98 el nuevo valor de la casilla será **v[i]= (v[i-1]+v[i]+v[i+1])/3**; para la casilla 0 se supondrá que su adyacente izquierda es la casilla 99 y para la casilla 99 se supondrá que su adyacente derecha es la casilla 0.

El procedimiento se aplicará durante 200 iteraciones y para optimizarlo se desea que **la mitad del array la procese un thread y la otra mitad otro thread.**

Las **operaciones de cálculo de los nuevos valores** se realizarán **en un array auxiliar** y sólo cuando los dos threads hayan terminado su iteración volcarán los valores de éste sobre el real para continuar con la siguiente iteración.

Por tanto el **procedimiento** que seguirá **cada thread** es:

1. **Copiar los nuevos valores** en el array auxiliar
2. Cuando haya terminado de copiarlos deberá **esperar a que termine el otro thread antes de volcar los datos del array auxiliar en el array real.** De esta forma no se modifican los datos de una iteración antes de que el otro utilice la casilla que tienen en común
3. Una vez copiados los datos en el array real deberá **esperar a que el otro thread termine también de copiar los datos auxilares sobre el array real** antes de proceder a la siguiente iteración del paso 1. Al igual que en el paso 2, hay que esperar para que las casillas comunes estén actualizadas.

A continuación se da la estructura básica del programa. **Se pide añadir la sincronización en los apartados donde se indica**. ¶ **y** • ¸ de los hilos 1 y 2·¸

```
#define TAM 100
#define NUMITER 200
pthread_attr_t attr;
pthread_t idth[2];
float v[TAM];

// ¶ AÑADIR LAS VARIABLES QUE SE NECESITEN

void rellenarArray(){  //Dependiente del programa. No lo tiene que rellenar el alumno  }

void *hilo0(void *num) {
  int i,j;
  float vaux[TAM/2];
        • Hilo 0/AÑADIR LAS VARIABLES QUE SE NECESITEN

  for (j=0; j<NUMITER; j++) {
         vaux [0]= (v[TAM-1]+v[0]+v[1])/3;
         for (i=1; i<TAM/2 ; i++)
          vaux [i]= (v[i-1]+v[i]+v[i+1])/3;

        ¸Hilo 0//REALIZAR LAS OPERACIONES DE SINCRONIZACIÓN NECESARIAS Y LA
        COPIA DE LOS DATOS AL ARRAY REAL
        }

  pthread_exit(0);

}
```

```
void *hilo1(void *num) {
   int i,j;
   float vaux[TAM/2];
```

- **Hilo 1**//AÑADIR LAS VARIABLES QUE SE NECESITEN

```
   printf ("Hilo 1\n");

    for (j=0; j<NUMITER; j++) {

         for (i=TAM/2; i<TAM-1 ; i++)
             vaux [i-TAM/2]= (v[i-1]+v[i]+v[i+1])/3;
          vaux [TAM/2-1]= (v[TAM-2]+v[TAM-1]+v[0])/3;
```

Hilo 1//REALIZAR LAS OPERACIONES DE SINCRONIZACIÓN NECESARIAS Y LA COPIA DE LOS DATOS AL ARRAY REAL

```
        }

     pthread_exit(0);

}

int main(){

   int i;
   rellenarArray();
  pthread_mutex_init (&mtx1, NULL);
   pthread_mutex_init (&mtx2, NULL);
   pthread_attr_init(&attr);
   pthread_create(&idth[0],&attr,hilo0,NULL);
   pthread_create(&idth[1],&attr,hilo1,NULL);
   for (i=0; i<2; i++)
       pthread_join(idth[i],NULL);
return(0);

}
```

Ejercicio 4 [2,5 puntos]:

Suponiendo que estamos en un sistema de ficheros UNIX., cuyos i-nodos dispone de:

- Punteros directos a bloques (16).
- Punteros indirecto simple (2).
- Punteros indirecto doble (2).

Además, el tamaño de bloque es de 1024 bytes y el tamaño de un índice es de 2 bytes. Suponga para todos los casos que sólo se encuentra precargado en memoria el i-nodo.

Responda a las siguientes preguntas:

a. ¿Cuantos accesos a disco son necesarios para leer completamente un fichero cuyo tamaño es de 10 MB? Asuma que una vez leído un bloque, este se encuentra en cache.

b. Considerando únicamente la estructura de i-nodos ¿Cuál es el tamaño máximo de un fichero en este sistema de ficheros?

c. ¿Cual es el tamaño máximo de un dispositivo de almacenamiento usando este sistema de ficheros?

d. ¿Qué ocurriría en este sistema de ficheros al aumentar el tamaño del bloque de datos? Explique las ventajas e inconvenientes de esta modificación.