SISTEMAS OPERATIVOS - CUESTIONES
13 de Septiembre de 2016

Nombre _____ DNI_____

Apellidos _____ Grupo _____

Cuestión 1. (1 punto)
Traducir la siguiente estructura de i-nodos a su correspondiente tabla FAT
Tabla de nodos-i

nodo-i	2	4	7	8	9	10	11	16
Tipo	D	D	D	F	F	F	F	D
Directo	1	2	4	5	10	15	9	3
Directo	null	null	null	6	12	8	null	null
Indirecto	null	null	null	null	null	7	null	null

Lista de bloques relevantes (los bloques que no aparecen aquí contienen datos o están vacíos)

1		2		3		4		7
.	2	.	4	.	16	.	7	18
..	2	..	2	..	4	..	2	19
home	4	carpeta	16	febrero.odt	8			
usr	7	tmp.txt	11	septiembre.txt	9			
				junio.odt	10			

Mapa de bits (el bit de más a la izquierda representa el bloque 1): 11111111110100100110.

En la siguiente figura se representa una tabla FAT. Al borde de sus entradas se ha escrito, como ayuda de referencia, el número correspondiente al bloque en cuestión. También se ha representado la entrada de cierto directorio. Como simplificación del ejemplo, suponemos que en cada entrada del directorio se almacena: nombre de fichero, tipo (F=archivo, D=directorio) y número del bloque inicial.

#Bloque	Índice	#Bloque	Índice
1	<EOF>	11	
2		12	
3		13	
4		14	
5		15	
6		16	
7		17	
8		18	
9		19	
10		20	

Nombre	Tipo	Nº Bloque
root	D	1
home		
usr		
carpeta		
tmp.txt		
febrero.odt		
septiembre.txt		
junio.odt		

Cuestión 2. (1 punto) Contestar a las siguientes preguntas sobre diferentes formas de planificar la ejecución de este conjunto de tareas.

Tarea	Llegada	CPU	E/S	CPU
T1	0	4	2	2
T2	2	5	1	1
T3	1	2	3	3
T4	0	6		

a) ¿Cuál será la última tarea en completar su ejecución si se aplica SJF expropiativo?
b) ¿Cuál será el tiempo total de ejecución con Round Robin y q=3?
Nota: marcar el tiempo de CPU coloreando y el de E/S rayando

SJF expropiativo

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T1																									
T2																									
T3																									
T4																									

Round Robin q=3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T1																									
T2																									
T3																									
T4																									

Cuestión 3. (1 punto) Responda brevemente a las siguientes preguntas:
a) ¿En qué directorio se alojan por convenio los ficheros de dispositivo en un sistema GNU/Linux?
b) Si se lista el directorio en cuestión con la orden ls -l tal como aparece más abajo, ¿qué representan los dos números separados por coma?
c) Si se desea instalar un módulo en el kernel con su código implementado en leds.c, y asociarlo a un fichero de dispositivo tipo carácter llamado leds, ¿qué comandos hay que utilizar y en qué orden? Se supone que el fichero Makefile está disponible.

```
$ ls -l
...
brw-r----- 1 root disk 3, 0 Nov 19 10:20 hda
crw-rw---- 1 root uucp 4, 64 Nov 19 10:20 ttyS0
crw-rw---- 1 root audio 14, 3 Dec 2 00:31 dsp
crw-rw-rw- 1 root root 1, 8 Nov 19 10:20 random
...
```

Cuestión 4. (1 punto) Un proceso en UNIX ejecuta el siguiente código. Explique qué es lo que hace el programa y si depende del orden en que se ejecuten los procesos. Indique cuál será el contenido del fichero "outfile.txt" al finalizar la ejecución.

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

int print_count=3;
char* text="1";

void *thread_function(void *arg) {
 int fd1;
 int i=0;
 int n=print_count;

 print_count--;
 fd1 = open("outfile.txt", O_WRONLY);
 lseek(fd1, 0, SEEK_SET);

 if (n<3)
   text="2";

 for (i=0;i<n;i++)
   write(fd1,text,strlen(text));

 close(fd1);
}

int main(int argc, char *argv[]) {
  pid_t pid;
  int stat, res;
  pthread_t a_thread, b_thread;
  void *thread_result;
  int fd1;

  fd1 = open("outfile.txt", O_CREAT | O_TRUNC | O_RDWR, S_IRUSR | S_IWUSR);
  write(fd1, "Init\n", 5);
  pid = fork();
  switch (pid) {
    case -1:
      write(fd1,"fork error!!\n",13);
      return 1;
    case 0:
       res = pthread_create(&a_thread,NULL, thread_function, NULL);
       write(fd1,"Waiting for child thread to finish...\n",38);
       res = pthread_join(a_thread, &thread_result);
       return 2;
    default:
       res = pthread_create(&b_thread,NULL,
             thread_function, NULL);

       write(fd1,"Waiting for parent thread to finish...\n",39);
       res = pthread_join(b_thread, &thread_result);

       wait(&stat);
       write(fd1,"parent process finished!!\n",26);
  }
  close(fd1);
  return 0;
```

Cuestión 5. (1 punto) Implemente las operaciones cond_wait() y cond_broadcast() de una variable condicional (my_cond_t) usando semáforos generales. Especifique también el tipo de datos my_cond_t e implemente su función de inicialización cond_init().

```
typedef struct {

} my_cond_t;
```

```
int cond_init(my_cond_t* c) {

}
```

```
int cond_wait(my_cond_t *vc, mutex_t* mtx) {

}
```

```
int cond_broadcast(my_cond_t* cc) {

}
```

Cuestión 6. (1 punto) Estudie el siguiente código que se ejecutará en un sistema Linux e indique si cada una de las afirmaciones posteriores son ciertas o falsas JUSTIFICANDO SU respuesta para cada una de ellas. Asuma que, tras alojarse el marco de pila de la función main(), el sistema sólo ha asignado una página para pila, una para código y una para datos globales (tamaño de página de 4 KB).

```
#include <stdio.h>

int len ;

int getNumberIter() {
  return 200;
}
```

```
int main() {
  int i;
  int M[128];
  int x,y;

  x = M[0];
  y =0;
  len = getNumberIter();
  for (i=1; i < len; i++) {
      y = x + M[i];
      M[i] = y;
  }
  return y;
}
```

a) El código compilará correctamente pero siempre dará una excepción por Violación de segmento al ejecutarlo porque accedemos a elementos de M no reservados.
b) El código compilará correctamente. En ejecución, es posible pero no seguro que haya excepción por Violación de segmento. Si no se produce la excepción se corromperán datos de la pila pero la ejecución continuará.
c) El código compilará correctamente. En ejecución, se corromperán datos de la pila y, quizás, de otras regiones de memoria (como el heap, código…)
d) Podemos asegurar que se producirá una excepción si la función getNumberIter() devuelve un número mayor de 4096.

SISTEMAS OPERATIVOS - PROBLEMAS
13 de Septiembre de 2016

Nombre _____ DNI_____

Apellidos _____ Grupo _____

Problema 1. (2 puntos) Un sistema de ficheros de un SO diseñado a partir de UNIX utiliza bloques de disco de 32 bytes de capacidad. Para el direccionamiento de estos bloques se utilizan punteros de 8 bits. Cada nodo-i tiene 2 punteros de direccionamiento directo y 1 puntero indirecto simple. Parte del contenido del sistema de ficheros está indicado a continuación:
Mapa de bits (de bloques de datos. '1' es ocupado. El primer índice se corresponde con el bloque 0): 0011 1111 110 0000 0000...0
Tabla de nodos-i (el nodo-i 2 es la raíz del árbol)

nodo-i	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Enlaces	NA	NA	1	NA	NA				
Tipo F/D/S	D	D	F	D	D				
Tamaño (B)	32	14	70	7	7				
Directo 1	6	7							
Directo 2	nil	nil		nil	nil				
Indirecto	nil	nil		nil	nil				

Lista de bloques relevantes:

Bloque 6	
.	2
..	2
opt	3
tmp	5
var	6

Bloque 7	
.	
..	2
troll	4

Bloque 8	
.	5
..	2

Bloque 9	
.	
..	2

a. **(1 puntos)** Dibuje el árbol de directorios y complete las estructuras del sistema de ficheros razonando en cada caso lo que se hace.

b. **(0.5 puntos)** Si a partir de ese sistema inicial se ejecutan los siguientes comandos:

```
> cd /
> ln -s /opt/troll /tmp/ogre
> ln /opt/troll /tmp/krampus
> rm /opt/troll
```

Indique razonadamente cuál será el estado final de las estructuras del sistema operativo.

c. **(0.5 puntos)** Considere el siguiente código e indique, para cada llamada al sistema marcada en negrita, el resultado que producen (indicando si habrá error) y cómo afecta al sistema de ficheros anterior (qué partes se modifican, cómo…), teniendo en cuenta los cambios en el sistema de ficheros producidos por la ejecución de los comandos del apartado b:

```
char buf[] = "I am a beast\n";
int fd = open("/opt/troll",O_RDWR | O_CREAT);
write(fd, buf, strlen(buf));
close(fd);
```

Explique razonadamente lo que se mostrará por pantalla si tras la ejecución de este código ejecutamos desde el terminal la orden cat /tmp/ogre.

Problema 2. (2 puntos) Se desea implementar un simulador de una carrera de relevos 4x100, donde compiten 4 equipos de corredores, formados por 4 miembros cada uno. Cada equipo de corredores está identificado mediante un número (del 0 al 3) y cada corredor también tiene asociado un identificador único (del 0 al 15). Los corredores del primer equipo tienen asociados identificadores del 0 al 3, los del segundo del 4 al 7, y así sucesivamente.

Para proporcionar mayor flexibilidad, el simulador se implementa como un programa multihilo, donde cada uno de los corredores está representado por un hilo independiente. El programa principal del simulador se limita a lanzar los 16 hilos que representan a los corredores. Cada corredor se comporta del siguiente modo:

```
void Corredor(int id_corredor){
   start_race(id_corredor);
   run(id_corredor);
   notify_arrival(id_corredor);
}
```

Cuando un corredor comienza su ejecución invoca la función start_race() que bloqueará al corredor hasta que sea su turno. Todos los corredores se bloquearán hasta que los demás corredores hayan invocado la función start_race(). Además, para los corredores que no salgan en primer lugar, la función bloqueará al corredor hasta que el corredor que le antecede en el equipo haya completado su tramo de la carrera (le haga entrega del testigo).

La función run() emula el comportamiento del corredor en base a sus características físicas, y se proporciona ya implementada. Esta función retorna cuando el corredor ha completado el tramo de los 100 metros que le corresponde. Al completar su tramo, el corredor procede a notificar al sistema invocando notify_arrival(), que se encarga de despertar al siguiente corredor en espera (si no es el último). En el caso de que sea el último corredor del equipo, esta función imprimirá por pantalla el identificador del equipo junto al turno relativo de llegada a la meta (1,2,3 o 4).

El programa multihilo debe garantizar lo siguiente:
No puede haber dos corredores del mismo equipo ejecutando la función run() simultáneamente
Un corredor debe comenzar su carrera lo antes posible, en cuanto se cumplan las restricciones anteriormente citadas

Implemente las funciones start_race() y notify_arrival()utilizando mutexes, variables condicionales y otras variables compartidas.

Nota: Además de proporcionar el código de estas funciones, se ha de describir claramente para qué sirve cada variable/recurso de sincronización utilizado, así como su valor inicial.