SISTEMAS OPERATIVOS - CUESTIONES
1 de Febrero de 2018

Nombre _____ DNI _____

Apellidos _____ Grupo _____

Cuestión 1. (1 punto) Sea un sistema de ficheros basado en nodos-i con un tamaño de bloque de 16 Bytes, con el siguiente contenido:

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): 11111111110100100110000000000.

Indicar los cambios producidos en las tablas al realizar las siguientes operaciones:

a) `$ echo "En un lugar de la Mancha, de cuyo nombre no quiero acordarme..." >> /home/carpeta/quijote.txt`

NOTA: "En un lugar de la Mancha, de cuyo nombre no quiero acordarme..." ocupa 64 Bytes

b) `$ ln /home/carpeta/quijote.txt /usr/link_quijote`

c) `$ cp /usr/link_quijote /usr/copia_link`

Cuestión 2. (1 punto) Un módulo del kernel con fichero de dispositivo `/dev/print_log` tiene las siguientes funciones asociadas a la apertura, cierre, lectura y escritura de su fichero de dispositivo:

```
static char msg[]="Print log module V0.1\n"
static int Device_Open = 0;

static int device_open(struct inode *inode, struct file *file){
    if(Device_Open++)  return -EBUSY;
    msg_Ptr = msg;
    try_module_get(THIS_MODULE);
    return SUCCESS;
}
static int device_release(struct inode *inode, struct file *file){
    Device_Open--;
    module_put(THIS_MODULE);
    return 0;
}

static ssize_t device_read(struct file *filp, char *buffer, size_t length, loff_t * offset){
    int bytes_to_read = length;
    if (*msg_Ptr == 0)
        return 0;
    if (bytes_to_read > strlen(msg_Ptr))
        bytes_to_read=strlen(msg_Ptr);
    if (copy_to_user(buffer,msg_Ptr,bytes_to_read))
        return -EFAULT;
    msg_Ptr+=bytes_to_read;
    return bytes_to_read;
}
static ssize_t device_write(struct file *filp, const char *buff, size_t len, loff_t * off){
    printk(KERN_INFO "%s\n", buff);
    return len;
}
```

Supongamos que ejecutamos las siguientes órdenes en línea de comandos:

1. cat /dev/print_log
2. echo “Mensaje de Log” > /dev/print_log

Conteste razonadamente a las siguientes preguntas:

a. ¿Qué mensajes aparecerán por pantalla? ¿y en el fichero de log?

b. ¿Hay algún error en las funciones que pueda provocar que el módulo falle al hacer las operaciones de lectura o escritura? Si es así indique cómo corregirlo.

Cuestión 3. (1 punto) Considere un sistema monoprocesador con una política de planificación de procesos round-robin con quanto 3. Inicialmente, hay 3 procesos en la cola del planificador: P1, P2 y P3 (en este orden). Estos procesos se ejecutan de forma indefinida siguiendo los siguientes patrones de CPU y E/S:

P1 (3-CPU, 5-E/S), P2 (7-CPU, 3-E/S) y P3 (4-CPU, 4-E/S)

Complete el diagrama de planificación que se muestra a continuación mostrando el estado de los procesos durante las 20 primeras unidades de tiempo. Indicar también el tiempo de espera de cada proceso y el porcentaje de utilización de la CPU durante las 20 primeras unidades de tiempo. Al rellenar el diagrama, se ha de usar el siguiente convenio para representar los estados de las tareas:

- En ejecución: marcar con "**X**"
- Bloqueado por E/S: marcar con **"--"**
- Listo para ejecutar: marcar con **"O"**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P1																				
P2																				
P3																				

Cuestión 4. (1 punto) Considere una aplicación en la que cuatro hilos concurrentes acceden y modifican dos variables globales compartidas x e y. Para proteger el acceso a las mismas, pero incrementando el grado de paralelismo potencial dentro de las secciones críticas, se han utilizado los semáforos s1 y s2 que han sido inicializados a 1, tal y como se describe en el siguiente código:

```
void thread1(void){
   while (1) {
     wait(s1);
     wait(s2);
     x=x-1;
     y=y+x;
     signal(s2);
     signal(s1);
  }
}
```

```
void thread2 (void) {
   while (1) {
     wait(s2);
     wait(s1);
     x=x-2;
     y=y+2*x;
     signal(s1);
     signal(s2);
  }
}
```

```
void thread3 (void) {
   while (1) {
     wait(s1);
     x=x-3;
     signal(s1);
   }
}
```

```
void thread4 (void) {
   while (1) {
     wait(s2);
     y=y+2;
     signal(s2);
  }
}
```

Contestar razonadamente a la siguiente pregunta. ¿Puede producirse algún tipo de problema al entrelazarse de forma concurrente la ejecución de los 4 threads? En caso afirmativo indicar por qué y proponer una implementación alternativa correcta utilizando dos semáforos.

Cuestión 5. (1 punto) Un proceso en UNIX ejecuta el siguiente código:

```
int fd1, fd2;
char buf[16];
int numB = 0;

void * leer ( void * arg ) {
  int fd3=open("prueba.txt", O_RDONLY);
  while((read(fd3,buf, 1))!=0)
    numB = numB+1;
  close(fd3);
  return NULL;
}
void * escrb ( void * arg ) {
  buf = "Final de fichero";
  int fd3=open("prueba.txt", O_WRONLY);
  lseek(fd3, 0, SEEK_END)
  write(fd3,buf,16);
  close(fd3);
  return NULL;
}

int main(void) {

   int PIDhijo=0;

   fd1=open("prueba.txt", O_RDWR | O_CREAT);

   PIDhijo = fork();

  if (PIDhijo == 0) {
    fd2=open("prueba.txt", O_RDONLY);

    pthread_create(& th1, NULL, leer, NULL);
    pthread_create(& th2, NULL, escrb, NULL);
    pthread_join ( th1 , NULL );
    pthread_join ( th2 , NULL );

    while((read(fd2,buf, 1))!=0)
      numB = numB+1;

    close(fd2);
    exit(0);
  }
  else {
    wait(&child_status);
    printf ("El numero de bytes es: %d", numB);
    if (close(fd1)!=0)
      printf("Error al cerrar fichero prueba");
    exit(0);
  }
  return 0;
}
```

Contesta las siguientes preguntas:

a) Indicar el valor de las diferentes tablas asociadas a la gestión de ficheros antes de ejecutarse cada uno de los close(), y sabiendo que los valores almacenados justo después de invocar fork() son los siguientes (suponer que cada fichero nuevo se almacena en filas consecutivas de la TFA). Añadir más tablas y/o filas si fuera necesario. Sólo en los casos en los que pudiera existir más de una solución correcta dependiendo de la planificación de procesos e hilos, indicad el orden de ejecución utilizado.

TDDA[1] Padre

DF	Ind. TFA
0	230
1	564
2	28
3	12
4	
5	

TDDA Hijo

DF	Ind. TFA
0	230
1	564
2	28
3	12
4	
5	

Tabla intermedia de posiciones (TFA)

	Pos L/E	Num nodo-i	Perm.	Cont. refs
...	...	...	...	...
12	0	57	R/W	2
13				
14	...	...	...	...
...	...	...	...	...

Tabla nodos-i

Nodo-i	Cont.
….	…..
Nodo-i 57	
...	….

b) ¿Qué valor imprime el printf del padre?

NOTA: Originalmente el fichero prueba.txt tiene almacenado “Este fichero es de prueba”

[1] Tabla de descriptores de ficheros (numéricos) abiertos por el proceso

Cuestión 6. (1 punto) Considerar un sistema que implementa gestión de memoria virtual con paginación por demanda (con páginas de 8 KBytes), direcciones de 64 bits y sin preasignación de la zona de intercambio (swap). Sobre este sistema se quiere ejecutar un programa cuyo fragmento más relevante se indica a continuación:

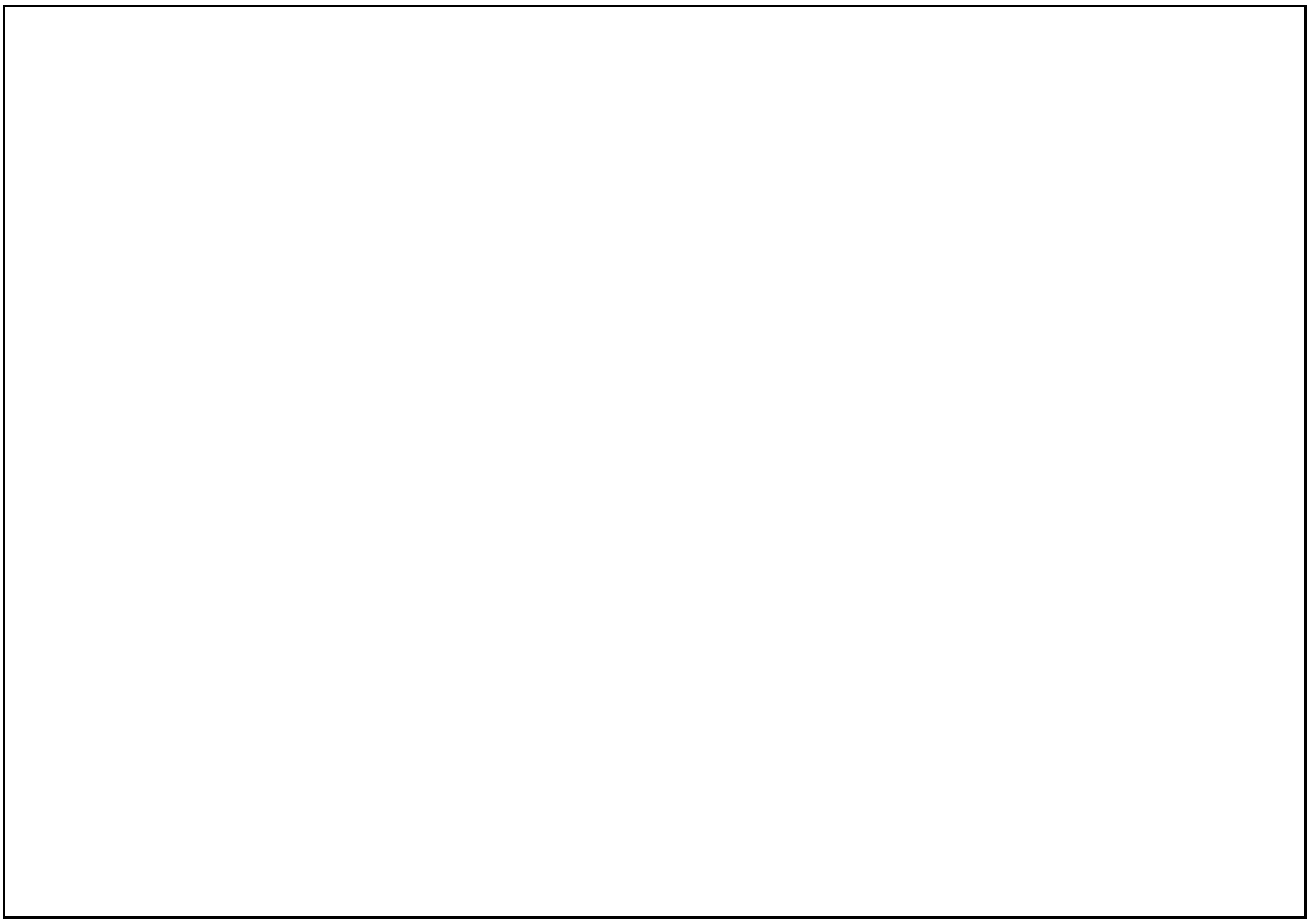
```
#define MYDATA "./mydata"
#define MYSIZE 8192

char* buf1;
char buf2[]="abcdefghijklmnopqrstuvwxyz";

int main(int argc, char* argv[]) {
   int fi, fo;
   char *datos;

   //Punto A
   buf1= (char *)malloc(MYSIZE*sizeof(char));

   fi=open(MYDATA,O_RDONLY);
   //Punto B

   if(read(fi,buf1,MYSIZE)==-1) {
      fprintf(stderr,"read error%d\n", errno);
      exit(EXIT_FAILURE);
   }
   //Punto C

  fo = open(MYDATA,O_WRDONLY);
  datos = (char *)mmap(NULL,1024,PROT_WRITE, MAP_SHARED,fo,0)
  //Punto D
  …
}
```

Por simplicidad, se realizarán las siguientes suposiciones:

- No se ejecuta ningún otro proceso y hay suficientes marcos de página libres en el sistema para alojar todas las páginas del proceso que sean necesarias.
- El contenido del fichero "./**mydata**" ocupa 2KB.
- Al generar el ejecutable se han enlazado todas las librerías de forma estática. El texto del programa tras la compilación ocupa 1 página y las librerías utilizadas 5 páginas.
- Al lanzar a ejecución el programa, la pila se encuentra inicializada y ocupa 512 bytes.
- Los buffers declarados y el código de la función main se encuentran alineados al comienzo de una página.

a) Indique las distintas regiones que conforman la imagen de memoria del proceso y su tamaño en páginas en los siguientes puntos: justo antes de que comience la ejecución de la función main, Punto A, Punto B, Punto C y Punto D.

b) Para los mismos puntos que el apartado anterior indicar (b.1) el número de marcos de página asignados al proceso, (b.2) el número de fallos de página que se han producido hasta el momento y (b.3) el número de páginas que han sido transferidas desde la memoria secundaria a la memoria principal hasta el momento.



SISTEMAS OPERATIVOS - PROBLEMAS
1 de Febrero de 2018

Nombre _____ DNI _____

Apellidos _____ Grupo _____

Problema 1. (2 puntos) Sean los fragmentos de programas de la tabla adjunta. Los programas, lanzados por el mismo usuario, se ejecutan de forma entrelazada de acuerdo a la secuencia establecida en dicha tabla. Supondremos que el fichero "doc.txt" existe en el nodo-i 13 y tiene un tamaño de 43 bytes (contiene 43 "x" seguidas), que el tamaño de bloque del sistema de ficheros (macro `TBLOQUE`) es de 1 KiB, y que ninguna de las instrucciones produce error. Además, supondremos que las direcciones de disco son de 16 bits, y que los nodos-i contienen 8 direcciones de disco para bloques de datos directos, una dirección de bloque índice indirecto simple y una dirección de bloque índice indirecto doble.

Orden	Proceso 1	Proceso 2
1	`fd = open("~/doc.txt", O_RDWR);`	
2	`write(fd, "145243", 4);`	
3		`fd = open("~/doc.txt", O_WRONLY);`
4		`lseek(fd, 3, SEEK_CUR);`
5	`lseek(fd, 50, SEEK_SET);`	
6		`write(fd, "7824", 4);`
7	`write(fd, "123434", 5);`	
8	`lseek(fd, 600*TBLOQUE, SEEK_SET);`	
9	`write(fd, "86", 2);`	
10		`close(fd);`

Se pide:

a) ¿Cuál es el tamaño máximo de un fichero en este sistema según la organización de nodos-i? Si para el Puntero de L/E de la la tabla intermedia de posiciones (TFA) se utiliza un entero de 2B, ¿cuál es el tamaño máximo del fichero según este dato?, ¿acarrean estos dos parámetros algún problema?

b) Mostrar el contenido del fichero al final de la ejecución de ambos procesos, tras la fila 10.

c) Indicar el número de bloques de disco (incluyendo también bloques de índices) que ocupa el fichero al final de la ejecución de ambos fragmentos de programa. Dibujar un esquema del nodo-i del fichero en el que se muestre únicamente el estado de los punteros y la relación de éstos con los bloques de disco asociados al fichero.

d) ¿Ocuparía el fichero al final de la ejecución el mismo número de bloques que el calculado en el apartado c) en el caso en el que se hubiera empleado un sistema de ficheros FAT con bloques de 1KiB? Justifique la respuesta.

Problema 2. (2 puntos) Se desea implementar un simulador de una carrera de natación de relevos 4x100m estilo libre, donde compiten 8 equipos de nadadores, formados por 4 miembros cada uno. Cada equipo de está identificado mediante un número (del 0 al 7) y cada nadador también tiene asociado un identificador único (del 0 al 31). Los nadadores del primer equipo tienen asociados identificadores del 0 al 3, los del segundo del 4 al 7, y así sucesivamente.

Para mejorar la escalabilidad del programa, el simulador se implementa como un programa multihilo, donde cada uno de los nadadores está representado por un hilo independiente. El programa principal del simulador se limita a lanzar los 32 hilos que representan a los nadadores. Cada nadador se comporta del siguiente modo:

```
void Nadador(int id_nadador){
   begin_race(id_nadador);
   swim(id_nadador);
   notify_arrival(id_nadador);
}
```

Cuando un nadador comienza su ejecución invoca la función `begin_race()` que bloqueará al nadador hasta que sea su turno. Todos los nadadores se bloquearán hasta que el último haya invocado la función `begin_race()`. Además, para los nadadores que no salgan en primer lugar (*id_nadador mod 4 ≠ 0*) , la función bloqueará al nadador hasta que el que le antecede en el equipo haya completado sus 100m correspondientes (le haga entrega del testigo).

La función `swim()` emula el comportamiento del nadador en base a sus características físicas, y se proporciona ya implementada. Esta función retorna cuando el nadador ha completado su tramo de 100 metros. Acto seguido, el nadador procederá a notificar al sistema invocando `notify_arrival()`, que se encarga de despertar al siguiente nadador en espera (si no es el último). En el caso de que sea el último nadador del equipo, esta función imprimirá por pantalla el identificador del equipo junto al turno relativo de llegada a la meta (de 1 a 8).

El programa multihilo debe garantizar lo siguiente:

- No puede haber dos nadadores del mismo equipo ejecutando la función `swim()` simultáneamente
- Un nadador debe comenzar su carrera lo antes posible, en cuanto se cumplan las restricciones anteriormente citadas

Implemente las funciones `begin_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.