Apellidos:_____ **Nombre:**_____ **DNI:** _____

Sistemas Operativos – Grado Ingeniera Informática UDC. Enero 2017
Sólo puede usar lápiz, bolígrafo y calculadora.
Tiempo máximo para todo el examen: 3h

Parte Sistema Ficheros
(Se deben contestar correctamente todas las cuestiones de cada pregunta para puntuar la misma).
Puntuación preguntas: P1: 0.6, P2: 0.4, P3: 0.4, P4: 0.6

P1) Un sistema de archivos tipo UNIX system V tiene un tamaño de bloque de 2Kb, i-nodos con 12 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 8 bytes. Calcular cuántos bloques de disco en el área de datos son necesarios para representar el archivo *"datos"* cuando su tamaño es de 130 Mbytes + 3 Kbytes. Discriminar cuántos bloques son de datos y cuántos de índices.

Número de bloques de datos: 65 Kbloques + 2 = 66562

Número de bloques de índices: 263

P2) En ese sistema de archivos UNIX (tamaño de bloque de 2Kbytes), el superbloque mantiene un mapa de bits de inodos libres para determinar los inodos libres/ocupados de la lista de inodos. Ese mapa de bits, que es parte del superbloque, ocupa un total de 2 bloques, mientras que la lista de inodos en disco ocupa 2048 bloques. Calcular cuál es el tamaño en bytes de cada inodo.

Tamaño en bytes de un inodo: 128

Apellidos:_____ **Nombre:**_____ **DNI:** _____

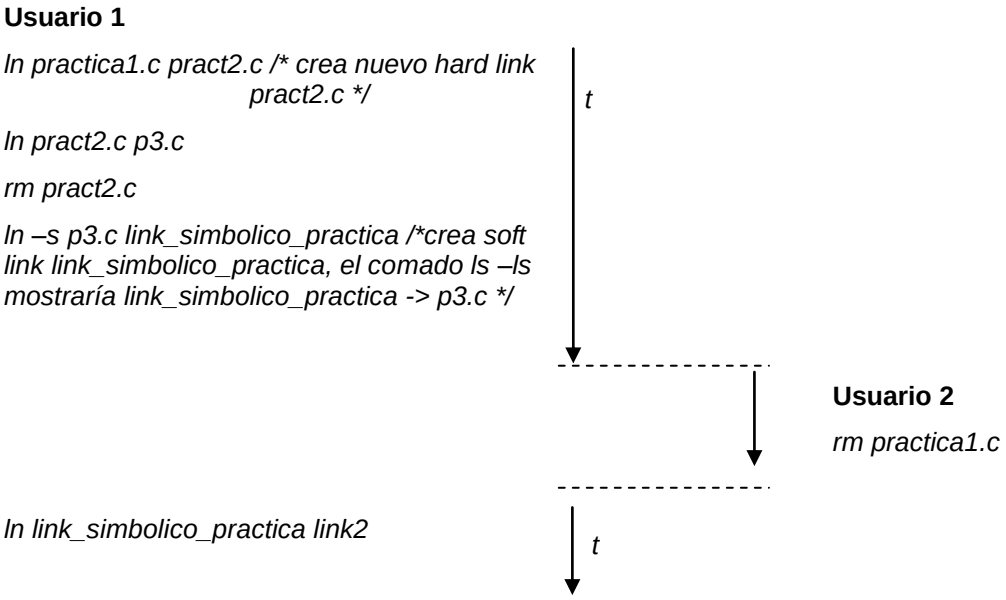
P3) Un proceso abre el archivo "/home/usr1/datos" en modo escritura, cuyo número de hard links es 1. El usuario efectivo del proceso coincide con el propietario del fichero, y tiene además los permisos de acceso al directorio *home* y *usr1*. Las cachés de datos e inodos están inicialmente vacías y la entrada *home* está en el tercer bloque del directorio raíz. Indicar lo siguiente referente al trozo de código:

```
struct stat buf;
char buffer[1024];
chmod("/home/usr1/datos", 0644);
if (lstat ("/home/usr1/datos", &buf)!=-1){
        printf("%s", convertir_presmisos(buf.st_mode)); /* se convierten los permisos a formato
                                                           rwxrwxrwx y se imprimen*/
        int fd=open("/home/usr1/datos", O_WRONLY); /* es la primera apertura de fichero del
                                                      proceso */
        lseek(fd, 1024, SEEK_SET);     /* SEEK_SET indica que el desplazamiento
                                          se considera a partir del origen del fichero) */
        write(fd, buffer, 512);
        close(fd);
        }
```

A. ¿Cuál es el valor asignado al descriptor de fichero *fd*?: 3
B. La llamada *write* escribe los bytes 1024 a 1536 del fichero. Cierto/Falso: Cierto
C. Los datos de la variable *buffer* en la escritura con *write* se guardan en el Buffer Cache y se actualizan inmediatamente a disco. Cierto/Falso: Falso
D. Si es un sistema de ficheros con registro, no existen problemas de pérdida de información del Buffer Cache ante caídas de alimentación. Cierto/Falso: Falso
E. Indica los permisos del fichero impresos en formato *rwxrwxrwx*: rw-r--r--

Apellidos:_____ **Nombre:**_____ **DNI:** _____

P4) La siguiente secuencia de comandos se ejecuta por parte de 2 usuarios del mismo grupo en el orden temporal especificado. El fichero "*practica1.c*" corresponde al inodo 34786 y tiene inicialmente número de (hard) links igual a 1 y un tamaño de 2Kbytes. Ambos usuarios tienen acceso al directorio que contiene el fichero *practica1.c*, incluyendo el permiso "w" en ese directorio tanto para el propietario del directorio como para el grupo de ese directorio (que es el mismo grupo al que pertenecen ambos usuarios).



A. ¿Cuál es el número de hard links final del inodo 34786?, después de ejecutar todos los comandos por parte de ambos usuarios: 1

B. Al ejecutar *rm practica1.c* por parte del usuario 2, se borran los datos del fichero en el área de datos pero su inodo no se libera (continúa como asignado en la lista de inodos en el disco). Indicar si es cierto o falso. Cierto/Falso: Falso

C. Al ejecutar *rm practica1.c* por parte del usuario 2, se borra la entrada de directorio con nombre "practica1.c". Indicar si es cierto o falso. Cierto/Falso: Cierto

D. ¿Cuál es el tamaño del fichero *link2*?: 4 bytes (indicar unidad: bytes/Kbytes/Mbytes)

Apellidos:_____Nombre:_____ DNI: _____

Sistemas Operativos – Grado Ingeniera Informática UDC. Enero 2017
Sólo puede usar lápiz, bolígrafo y calculadora.
Tiempo máximo para todo el examen: 3h

Parte Memoria.

1. (0.4 puntos) Considere el siguiente código C con tres implementaciones distintas de la función readdata()

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char *readdata()
{
   ---------
}
int main(int argc, char *argv[])
{
   char *name, *surname;

   printf("Enter name and press Enter key (less than 20 chars):");
   name = readdata();
   printf("Enter surname and press Enter key (less than 20 chars):");
   surname = readdata();

   printf("\nName:%sSurname:%s", name, surname);

   return 0;
}
```

Implementación 1:	Implementación 2:	Implementación 3:

char *readdata()
{
  char data[30];
  fgets(data, 30, stdin);
  return data;
}
	char *readdata()
	{
	  static char data[30];
	  fgets(data, 30, stdin);
	  return data;
	}
		char *readdata()
		{
		  char *data = malloc (40);
		  fgets(data, 30, stdin);
		  return data;
		}

Indique la respuesta correcta y explíquelo. La explicación debe ser correcta y suficiente.

a) Las 3 implementaciones son correctas b) Las 3 implementaciones son incorrectas
c) Sólo la 1 es correcta d) Sólo la 2 es correcta e) Sólo la 3 es correcta
f) Sólo la 1 es incorrecta g) Sólo la 2 es incorrecta h) Sólo la 3 es incorrecta

Un implementación es correcta si produce un ejecutable que permite leer el nombre y apellido de un sujeto e imprimirlos por pantalla, suponiendo que el usuario sigue siempre las instrucciones que se le indican. Los tres ejecutables que se corresponden con las tres variantes, ninguno da error en tiempo de ejecución.

Respuesta: __e__
Explicación: En la 1, la var data es local con lo cuál estará en el stack cuando se llama la función, pero en la segunda llamada a readdata() la zona de memoria del stack para la primera llamada se reescribe, con lo cual no es correcto, porque se reescribe el valor apuntado por la variable name antes de hacer uso en printf. Con la implementación 2, a pesar de que la var data es estática y reside en el segmento de datos, el problema es parecido porque la segunda llamada a readdata() reescribe el valor de la primera, en esa zona de las var estáticas del segmento de datos. La 3 es correcta porque la var data apunta al heap y cada malloc obtiene una zona de memoria distinta en el heap que es a la que apuntan las variables name y surname del main().

2. (0.3 puntos) Considere un sistema de gestión de la memoria básico como el estudiado en clase con un registro base (o de reubicación) y un registro límite proporcionados por el hardware.
Indique la respuesta correcta y explíquelo. La explicación debe ser correcta y suficiente.
a) No permite protección de la memoria
b) No permite relocalización de direcciones
c) Permite que un programa tengas varios segmentos (más de 1) de memoria
d) No presenta fragmentación externa ni fragmentación interna
e) Permite multiproceso
f) Ninguna de la anteriores es cierta

Respuesta: ___e_
Explicación:
Un proceso en ejecución sólo puede tener un segmento y los valores de los registros base y límite definen la posición de ese segmento. Esos valores se pueden guardar en el PCB (Process Control Block) para que ese proceso deje la CPU y otro proceso la tome cargándose los registros con los valores almacenados en el PCB de este último, de esta forma permitiendo el multiproceso. El esquema permite protección de la memoria y relocalización dinámica y en los sistemas con segmentación la fragmentación externa es una de las desventajas.

3. (0.3 puntos)
a) Un proceso en modo usuario puede modificar las entradas de las tablas de páginas de cualquier otro proceso en modo usuario.
b) Un proceso en modo usuario puede modificar las entradas de cualquier (de cualquier nivel) tabla de páginas de ese mismo proceso, pero no las entradas de tablas de páginas de otros procesos.
c) Un proceso en modo usuario sólo puede modificar las entradas de tablas de páginas de último nivel de ese mismo proceso
d) Un proceso en modo usuario sólo puede modificar las entradas de la tabla de páginas de ese mismo proceso que apuntan al heap (datos dinámicos) o stack (pila) del proceso.
e) Un proceso en modo usuario no puede modificar las entradas de la tablas de páginas de ese mismo proceso.
Indique la respuesta correcta y explíquelo. La explicación debe ser correcta y suficiente.
Respuesta: ___e_
Explicación: Sólo el kernel puede modificar las entradas de las tablas de páginas de los procesos, en caso contrario un proceso pudiera acceder a la memoria física de otros proceso o del kernel.

4. a) (0.4 puntos) Considere un sistema con 16Gbytes de memoría física, páginas de 4Kbytes y entradas en las tablas de páginas de 4 bytes. Con un nivel la TP ocupa una pagina. Con varios niveles la TP raíz o la de un nivel que no es el último, apunta a páginas que contienen TPs. ¿Cual es el tamaño en bytes del espacio virtual direccionable si se usa tablas de paginas de 1, 2 o 3 niveles?
De la respuesta en bytes y a continuación los cálculos
Con la TP de un nivel: _____ $2^{\wedge 22}$ bytes
Con TPs de dos niveles: _____$2^{\wedge 32}$ bytes
Con TPs de tres niveles: _____$2^{\wedge 42}$ bytes

4KB = $2^{\wedge 12}$ bytes, por tanto cada TP tiene $2^{\wedge 12}$ / $2^{\wedge 2}$ = $2^{\wedge 10}$ entradas, y cada entrada apunta a una página de $2^{\wedge 12}$ bytes por tanto con una TP de un nivel se direccionan $2^{\wedge 10}$ x $2^{\wedge 12}$ = $2^{\wedge 22}$ bytes

Con TPs de dos niveles, la primera apunta a $2^{\wedge 10}$ tablas de páginas de segundo nivel y cada una de segundo nivel puede direccionar $2^{\wedge 22}$ bytes, por tanto se pueden direccionar $2^{\wedge 10}$ x $2^{\wedge 22}$ = $2^{\wedge 32}$ bytes

Con TPs de tres niveles, la primera apunta a $2^{\wedge 10}$ tablas de páginas de segundo nivel y cada una de segundo nivel puede direccionar $2^{\wedge 32}$ bytes, por tanto se pueden direccionar $2^{\wedge 10}$ x $2^{\wedge 32}$ = $2^{\wedge 42}$ bytes

b) (0.3 puntos) ¿Cuál es el número mínimo de bits necesarios en cada entrada de una tabla de páginas para poder direccionar toda la memoría física?
De la respuesta en byte y a continuación los cálculos
Número de bits necesarios: _____22 bits

16GB = $2^{\wedge 34}$ bytes, ya que las páginas físicas de 4KB, es un total de $2^{\wedge 34}$ / $2^{\wedge 12}$ = $2^{\wedge 22}$ páginas. Por tanto se necesitan 22 bits en la entrada de tabla de páginas para indicar la página física.

c) (0.3 puntos) Una vez calculado b), ¿pueden los bits restantes de la entrada de tabla de páginas usarse para (en cada item escriba SI/NO, no hace falta justificar en este caso, pero tienen que estar todos correctos para puntuar este

apartado)?
-para indicar si la página es de sólo lectura: _____SI
-para indicar si la página tiene permiso de ejecución: _____SI
-para indicar el offset dentro de la página: _____NO
-para indicar el número de página lógica: _____NO

El offset y el número de página lógica vienen dados en la dirección lógica, no en las entradas de la tabla de páginas

Sistemas Operativos

Grado en Informática. Enero 2017

Procesos

1. **(1 punto)**Se muestra la planificación de cuatro procesos con distintos algoritmos.

Alg1	A	A	A	A	A	A	A	B	B	B	C	D	D	D	D
Alg2	A	A	B	B	C	A	A	D	D	B	A	A	D	D	A
Alg3	A	A	A	A	A	A	A	C	B	B	B	D	D	D	D
Alg4	A	B	C	B	B	D	D	D	D	A	A	A	A	A	A

Identificar los algoritmos y estimar la duración de las ráfagas de CPU, asi como los instantes de llegada. Si se considera necesario añadir más información pueden usarse las columnas libres

Proceso	Ráfaga	Llegada		
A	7	0		
B	3	1		
C	1	2		
D	4	3(4)		

Rellenar la siguiente tabla con los tiempos de retorno y espera para cada proceso en los distintos algorimos. Calcular tambien los tiempos de retorno y espera promedio. Indicar de qué algoritmos se trata

	Alg1=FCFS	**Alg2=RR(q=2)**	**Alg3=SJF**	**Alg4=SRTF**
Tiempo retorno A	7	15	7	15
Tiempo retorno B	9	9	10	4
Tiempo retornoC	9	3	6	1
Tiempo retorno D	12(11)	11(10)	12(11)	6(5)
T_r promedio	9.25 (9)	9.5(9.25)	8.75(8.5)	6.5 (6.25)
Tiempo espera A	0	8	0	8
Tiempo espera B	6	6	7	1
Tiempo espera C	8	2	5 0	
Tiempo espera D	8(7)	7(6)	8(7)	2(1)
T_w promedio	5.5(5.25)	5.75(5.5)	5 (4.75)	2.75(2.5)

Si asumimos que cuando un proceso sale de la CPU en el mismo instante en que llega otro nuevo, es el nuevo el que se coloca delante en la cola de listos, entonces sería admisible considerar que el proceso D llega en el instante 4 (esa solución es la que se muestra entre paréntesis). SJF y SRTF tambien podrian ser considerados como planificaciones por prioridades no apropiativas y apropiativas respectivamente, poniendo las prioridades adecuadas

Viendo RoundRobin,vemos que D NO PUEDE haber llegado en el instante 5.

Tiempo de retorno se define como el tiempo transcurrido desde que un proceso llega hasta que termina y el tiempo de espera es el tiempo de retorno menos el tiempo de servicio, que al haber una sola ráfaga coincide con la duración de la ráfaga. Comenzar la numeración en 0 ó en 1 no cambia los valores

2. **(0.5 puntos)** Considerar el siguiente código en C (Tiene todos los include necesarios y compila correctamente)

```
main(int argc, char * argv[])
{
  unsigned long veces=4000000000;
  unsigned long i;
  pid_t pid;

  pid=fork();
  for (i=0; i<veces; i++)
      if ((pid=fork())==0)
           execl("./a.out","a.out","20",NULL);
      else
           execl("./a.out","a.out","2",NULL);
}
```

Ademas del primer proceso creado para ejecutarlo, ¿Cuantos procesos se crearán el los primeros 60 segundos? (*sleep(n)* deja un proceso en espera durante n segundos y despreciaremos el tiempo empleado en las llamadas fork() y exec(), y en las operaciones con enteros frente a segundos completos de espera). Suponemos *./a.out* existe, es un ejecutable válido y su código es:

```
main(int argc, char * argv[])
{
  int n;

  n=atoi(argv[1]);
  sleep(n);
}
```

Processos creados en los primeros 60 segundos: 3
Explicación: El proceso original (llamésmole P_0) crea un primer proceso hijo en la línea `"pid=fork();"` **(P_1) y ahora ambos (P_0 y P_1) ejecutarán la primera iteración del bucle con lo que en la línea** `"if ((pid=fork())==0)"` **cada uno de ellos creará un nuevo proceso. Los cuatro procesos (el original P_0, P_1 y los dos creados en la primera iteración del bucle) reemplazarán su código por el de** `a.out` **(P_0 y P_1 recibirán el parámetro 2 y los otros dos el parámetro 20) y ya no se crearán más procesos**

3. **(0.5 puntos)**Un sistema en tiempo real maneja 3 enventos periódicos con periodos $T_1 = 100ms$, $T_2 = 50ms$ y $T_3 = 200ms$, cuyos correspondientes tiempos de ejecución son $C_1 = 5ms$, $C_2 = 2ms$ y $C_3 = 100ms$. Queremos que maneje tambien un cuerto evento que ocurre cada 2 ms. ¿Cuál es el máximo tiempo de procesamiento de este evento que hace que el sistema siga siendo planificable?

   Para que un sistema en tiempo real que tiene que responder a una serie de flujos periódicos sea planificable, ha de verificarse

$$\sum_i \frac{C_i}{T_i} \leq 1$$

donde T_i es el periodo del flujo i y C_i el tiempo necesario para procesar el flujo i. En nuestro caso $\frac{5}{100} + \frac{2}{50} + \frac{100}{200} + \frac{x}{2} \leq 1$ y por tanto $x \leq 0.82ms$

ENTRADA/SALIDA opción: | A | B | Puntuación: |

EXAMEN DE SISTEMAS OPERATIVOS (Grado en Ing. Informática) 12/01/2017.

APELLIDOS Y NOMBRE: .....

En este apartado **DEBE elegir** entre contestar los ejercicios [A.1 y A.2] = **opción A**, o los [B.1 y B.2] = **opción B**. **Asegúrese de marcar** con una X la opción elegida en el encabezado.

NOTA: Si el/la alumno/a contestase a preguntas pertenecientes a ambas opciones, y no eligiese adecuadamente una opción en la casilla del encabezado, se considerará que la opción elegida es la **A** y **sólo** serán corregidas y puntuadas las respuestas correspondientes a la **opción A**.

[OPCIÓN A] (1.5 puntos)

A.1 **(1.0 puntos)** Asúmase que en un determinado instante la cola de peticiones de E/S para acceder a cilindros de un disco duro contenía las peticiones: $\langle 981, 601, 440, 300\rangle$. Desconocemos el cilindro X que estaba siendo accedido justo antes de estas peticiones, pero sabemos que X no estaba entre los cilindros de la lista de peticiones. Si sabemos que el orden de atención de dichas peticiones fue: $\langle 440, 300, 601, 981\rangle$. Indíquese si los siguientes algoritmos de planificación de accesos a disco pudieron ser utilizados (sí/no). En caso afirmativo, indíquese también UN posible número de cilindro (X) que podría estar siendo accedido antes de atender a dichas peticiones. Razone brevemente su respuesta.

algoritmo	**posible** (sí/no)	**posible cilind X**	**breve justificación de la respuesta**
SSTF:	sí	$[371, 520]$	$X < 521$ pues atiende 440 antes que 601. $X > 370$ pues atiende 440 antes que 300.
SCAN:	sí	$(440, 600]$	El ascensor está bajando y atiente 440, 300 al subir atiende 601, 981
C-LOOK:	no	$--$	Ascensor atiende peticiones al bajar (440, 300), así que debería atender 981 antes que 601
FCFS:	no	$--$	No se respeta orden de llegada de peticiones

A.2 **(0.5 puntos)** Un fichero de texto contiene `ABCDEFGHIJKLMNOPQRSTUVWXYZ`. Indíquese qué se imprime por pantalla al ejecutar el siguiente código. Justifíquese brevemente su respuesta.

```
l. 1: #include "includes.h"
l. 2: int main(){
l. 3:   char buf[10]="---------";
l. 4:   int fd = open("a.dat",O_RDONLY);  //existe en directorio actual
l. 5:   lseek(fd, 3, SEEK_SET);
l. 6:   pread(fd,buf,5,2);
l. 7:   read (fd,buf,2);
l. 8:   write(STDOUT_FILENO,buf,5);
l. 9:   exit(0);
l.10: }
```

Imprime: DEEFG
Justificación
En l.5, lseek establece el offset actual de lectura en el fichero a la posición 3 respecto al inicio.
En l.6, pread modifica buf[0..4] ← `CDEFG`, al leer 5 bytes a partir de la posicion 2 del fichero. Nótese que pread no usa ni modifica el offset actual en el fichero.
En l.7, read modifica buf[0..1] ← `DE`, pues lee 2 bytes desde la posición 3 del fichero, que era el offset actual dentro del fichero (tras el lseek de l.5.).

En l.8 write escribe en pantalla los 5 primeros bytes de buf; esto es: DEEFG

[OPCIÓN B] (1.5 puntos)

B.1 Describa brevemente la estructura (capas/*layers*) del software de entrada/salida.

ULS: User Level Software: Interfaz con el usuario y formateo de datos
DIL: Device Independent Layer: En esta capa tienen lugar todas las tareas de asignación de espacio, control de privilegios, uso de cache (si hay), manejo de un tamaño de bloque independiente del dispositivo, nombrado de dispositivos, etc.
DD : Device Driver: Es la capa de software que se comunica con el controlador de dispositivo y es la única pieza de software en todo el S.O. que lo hace. Por ello, este software puede implicar conocer propiedades particulares del dispositivo
IH: Interrupt Handler: Gestiona las interrupciones generadas por un dispositivo

B.2 Para cada una de la siguientes tareas, nombre la capa en la que se lleva a cabo.

- Convertir un número real a su representación en forma de *string* para ser escrito en la pantalla: ULS
- Enviar comandos al controlador SCSI para posicionar las cabezas del disco: DD
- Despertar a un proceso que estaba esperando por una entrada proveniente del teclado, una vez que la entrada se haya completado: IH
- Mantener un bitmap del espacio libre en algunos dispositivos de almacenamiento: DIL
- Implementar el algorithmo de planificación de disco C-Scan: DD
- LLevar cuenta de los ficheros abiertos por un proceso: DIL
- LLevar cuenta de todos los ficheros abiertos en el sistema: DIL
- Determinar en qué cilindro, cara y sectores de disco está un bloque: DD
- Determinar qué bloque de disco contiene un cierto offset de un fichero: DIL
- Chequear si un proceso que intenta abrir un fichero tiene los permisos necesarios DIL
- Chequear si un descriptor de fichero es válido para escritura: DIL
