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

Sistemas Operativos – Grado Ingeniera Informática UDC. Julio 2018
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: 0.5, P4: 0.5

P1) Un sistema de archivos tipo UNIX System V tiene un tamaño de bloque de 512 bytes, i-nodos con 12 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 4 bytes. Calcular el tamaño máximo de un fichero al utilizar los niveles de indirección simple, doble y triple.

Tamaño máximo usando puntero de indirección simple: 70Kbytes

Tamaño máximo usando puntero de indirección doble: 8 Mbytes + 70Kbytes

Tamaño máximo usando puntero de indirección triple: 1Gbyte + 8 Mbytes + 70Kbytes

P2) En ese sistema de archivos UNIX (tamaño de bloque de 512 bytes), el tamaño de un inodo es de 64 bytes. 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 8 bloques. Calcular cuántos bloques son necesarios para la zona de inodos en el disco (lista de inodos).

Número de bloques que ocupa la lista de inodos en disco: 4 Kbloques

P3) En ese sistema de archivos (P1), un proceso abre el archivo *"datos*" (tamaño 200 Mbytes) en modo lectura/escritura, *open ("datos", O_RDWR)*, que se encuentra en el directorio de trabajo del proceso (la apertura no retorna ningún error). Consideremos el siguiente trozo de código en esa apertura:

```
char c;
int fd=open("datos", O_RDWR); /* es la primera apertura de fichero por
                                 parte del proceso */
lseek(fd, 8388608, SEEK_SET);   /* SEEK_SET indica que el desplazamiento
                                   se considera a partir del origen del fichero
                                   (8388608= 2^23)  */
c=fgetc(fd);
printf ("La apertura del fichero retorna descriptor %d\n", fd);
close(fd);
```

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

Calcular cuántos bloques tendría el SO que leer del disco, discriminando entre bloques de datos y de índices, para realizar la operación anterior:
c=fgetc(fd);
suponiendo el buffer caché vacío. (Nota: *8388608*= 2^{23})

A. Número de bloques de índices que es necesario leer: 2

B. Número de bloques de datos que es necesario leer: 1

C. Al ejecutar *printf*, marcar cuál de estas opciones es cierta:
Se incrementa solo el tiempo de usuario del proceso
Se incrementa solo el tiempo de sistema del proceso
Se incrementan ambos (tiempo usuario y de sistema) X

P4) Supongamos las siguientes secuencias de comandos:

1. Se crean 3 *hard link* al fichero *"datos"* (cuyo número de inodo es 25634, tamaño 200 Mbytes y num. de *hard links*=1) en su mismo directorio:

ln datos hlink1

ln datos hlink2

ln datos hlink3

2. Posteriormente se crea un *soft link* al fichero *hlink3*:

ln -s hlink3 slink3 / crea soft link slink3 */*

/ el comando ls -l mostraría slink3 -> hlink3 */*

y otro un *soft link* al fichero *hlink2*:

ln -s hlink2 slink2 / crea soft link slink2 */*

/ el comando ls -l mostraría slink2 -> hlink2 */*

Contestar a lo siguiente:

A. Indicar el tamaño (en Gbytes, Mbytes o bytes) del fichero *slink3*: 6 bytes

B. Indicar cuál es el número de *(hard) links* del fichero *hlink2 (*después de crear todos los *hard links* en 1): 4

C. Una vez creados los ficheros *slink2* y *slink3*, al borrar el fichero "datos" (*rm datos*) se decrementa su número de *hard links*. ¿Se puede acceder al contenido del fichero con número de inodo 25634 a través del link simbólico *slink2?*

Sí

D. Tras la operación *rm datos,* ¿cuál es el número de *hard links* de *slink2*?: 1

Apellidos:_____Nombre:_____ DNI: _____

Sistemas Operativos – Grado Ingeniera Informática UDC. Julio 2018
Sólo puede usar lápiz, bolígrafo y calculadora. Smartphones apagados y en la mochila.
Tiempo máximo para todo el examen: **2h45m**

Parte Memoria

1. (0.6) Considere al siguiente código C. Al ejecutarse imprime por salida 6 direcciones. Indique sólo si cada dirección está en el segmento de datos estáticos, segmento de datos dinámicos (heap) o pila (stack). **No debe justificarse cada respuesta pero cada respuesta incorrecta o ambigua invalida una respuesta correcta.**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char *f1();
char *f2();
char **f3();
char **f4();
char **f5();
char **f6();

int main(void)
{
	char *b;
	char **c;

	b=f1();
	printf("\n [%p] \n", b);
	b=f2();
	printf("\n [%p] \n", b);
	c=f3();
	printf("\n [%p] \n", c);
	c=f4();
	printf("\n [%p] \n", c);
	c=f5();
	printf("\n [%p] \n", c);
	c=f6();
	printf("\n [%p] \n", c);

	return 0;

}

char *f1()
{
	char *s="I am s from f1";
	return s;
}
char *f2()
{
	char *s=NULL;
	s= (char *) malloc(100);
	strcpy(s, "I am s from f2");
	return s;
}
char **f3()
{
	char *s="I am s from f3";
	return &s;
}
char **f4()
{
	static char *s="I am s from f4";
	return &s;
}
char **f5()
{
	char *s=NULL;
	s= (char *) malloc(100);
	strcpy(s, "I am s from f5");
	return &s;

}
```

```
char **f6()
{
	static char *s=NULL;
	s= (char *) malloc(100);
	strcpy(s, "I am s from f6");
	return &s;
}
```

[0x4007bd] Datos estáticos

[0x1296010] Heap

[0x7ffc889bf298] Pila

[0x600c40] Datos estáticos

[0x7ffc889bf298] Pila

[0x600c50] Datos estáticos

2. (0.8) Considere un sistema de memoria virtual con tablas de páginas en dos niveles, direcciones virtuales de 32 bits donde los 12 bits menos significativos son para el offset, los 10 más significativos para la entrada en la TP de primer nivel y los 10 siguientes para la entrada en la TP de segundo nivel. Cada TP es de tamaño 1 página, que es lo habitual. Las direcciones físicas tienen los 20 bits más significativos para el número de página física y los 12 menos significativos para el offset. Las entradas en las TP tienen 32 bits cuyo significado se muestra en el orden de bits más significativos a menos significativos, y este mismo orden es de almacenamiento en memoria.

20 bits: número de página física
3 bits: reservados
1 bit: 0
1 bit: reservado
1 bit: dirty bit
1 bit: bit de acceso
1 bit: no cache
1 bit: write-through
1 bit: 1 user, 0 kernel
1 bit: 1 escritura, 0 read-only
1 bit: 1 página válida

Suponga que para un proceso el registro base a la TP de primer nivel contiene el valor (en hexadecimal) 0x00200000. En la tabla adjunto se muestran los contenidos de la memoria. El almacenamiento de datos muti-byte es big-endian. Indique el resultado en **hexadecimal** (o él tipo de error) de las siguientes operaciones de memoria a partir de los contenidos de memoria física que se adjuntan. Una operación Load devuelve un valor de 8 bits o un error. Una operación Store devuelve Ok o error. Errores posibles son página inválida, read-only o kernel-only. **Tanto el resultado final como los cálculos necesarios deben ser correctos para puntuar la pregunta.**

a) (0.4 puntos) **Load 0x0000003B Resultado:_____0xBB____**

número de página de primer nivel= 0
número de página de segundo nivel= 0
offset en la página = 0x03B

entrada 0 (primera entrada) TP de primer nivel =0x00100007
número página física TP segundo nivel= 0x00100
dir física base TP segundo nivel = 0x00100000
entrada 0 (primera entrada) TP de segundo nivel = 0x00001065, acaba en binario en 101: modo user, sólo lectura, página válida

número de página 0x00001
dir física base de la página 0x00001000
dir física base de la página + offset = 0x0000103B
contenido de la dir 0x0000103B = **0xBB**

a) (0.4 puntos) **Load 0x02003011 Resultado:_____0x28____**

0x02003, en binario 0000 0010 0000 0000 0011
número de página de primer nivel= en binario 0000 0010 00 = 0x08
número de página de segundo nivel= en binario 00 0000 0011=0x03
offset en la página = 0x011

entrada 0x08 (novena entrada) TP de primer nivel =0x00100007
número página física TP segundo nivel= 0x00100
dir física base TP segundo nivel = 0x00100000
entrada 0x03 (cuarta entrada) TP de segundo nivel = 0x00004007, acaba en binario en 111: modo user, escritura, página válida

número de página 0x00004
dir física base de la página 0x00004000
dir física base de la página + offset = 0x00004011
contenido de la dir 0x00004011 = **0x28**

3. (0.6 puntos) Considere cadena de referencia a páginas de un proceso que se muestra en la primera fila de la tabla Un sistema de memoria virtual asigna 6 páginas físicas a un proceso y usa un algoritmo de asignación y reemplazo que produce la siguiente asignación de páginas lógicas a páginas físicas con los fallos de página que se muestran. **Los apartados b) y c) no se puntúan si está mal el apartado a).**

1	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**5**	**1**	**7**	**2**	**3**	**3**	**6**
1	1	1	1	*	*	1	1	*	*	1	*	*	*	1
	2	2	2	*	*	2	2	*	*	2	*	*	*	2
		3	3	*	*	3	3	*	*	3	*	*	*	3
			4	*	*	4	4	*	*	4	*	*	*	4
						5	5	*	*	5	*	*	*	5
							6	*	*	7	*	*	*	6
F	**F**	**F**	**F**			**F**	**F**			**F**				**F**

a) (0.2) ¿De que algoritmo de reemplazo se trata? Pista: Puede tratarse de un algoritmo de reemplazo visto en clase u otro distinto. En cualquier caso debe indicar el principio de funcionamiento del algoritmo

El algoritmo usa una estrategia LIFO (último en entrar, primero en salir)

b) (0.2) ¿Es un buen algoritmo en términos de número de fallos de página que produce? Conteste SI/NO: _____
Y razone la repuesta.

NO. Porque va en contra del principio de localidad.

c) (0.2) ¿Es un buen algoritmo en términos del coste de la implementación? Conteste SI/NO: _____
Y razone la repuesta.

SI. Básicamente es el mismo coste que FIFO. El algoritmo sólo se invoca cuando hay un fallo de página, no necesita hardware adicional al de paginación y el SO sólo tiene que mantener una lista de referencias a páginas que gestiona con política LIFO.

Physical Memory [All Values are in Hexidecimal]

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
00000000	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
00000010	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D
....																
00001010	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
00001020	40	03	41	01	30	01	31	03	00	03	00	00	00	00	00	00
00001030	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF
00001040	10	01	11	03	31	03	13	00	14	01	15	03	16	01	17	00
....																
00002030	10	01	11	00	12	03	67	03	11	03	00	00	00	00	00	00
00002040	02	20	03	30	04	40	05	50	01	60	03	70	08	80	09	90
00002050	10	00	31	01	10	03	31	01	12	03	30	00	10	00	10	01
....																
00004000	30	00	31	01	11	01	33	03	34	01	35	00	43	38	32	79
00004010	50	28	84	19	71	69	39	93	75	10	58	20	97	49	44	59
00004020	23	03	20	03	00	01	62	08	99	86	28	03	48	25	34	21
....																
00100000	00	00	10	65	00	00	20	67	00	00	30	00	00	00	40	07
00100010	00	00	50	03	00	00	00	00	00	00	00	00	00	00	00	00
...																
00103000	11	22	00	05	55	66	77	88	99	AA	BB	CC	DD	EE	FF	00
00103010	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	00	67
...																
001FE000	04	15	00	00	48	59	70	7B	8C	9D	AE	BF	D0	E1	F2	03
001FE010	10	15	00	67	10	15	10	67	10	15	20	67	10	15	30	67
...																
001FF000	00	00	00	00	00	00	00	65	00	00	10	67	00	00	00	00
001FF010	00	00	20	67	00	00	30	67	00	00	40	65	00	00	50	07
...																
001FFFF0	00	00	00	00	00	00	00	00	10	00	00	67	00	10	30	65
...																
00200000	00	10	00	07	00	10	10	07	00	10	20	07	00	10	30	07
00200010	00	10	40	07	00	10	50	07	00	10	60	07	00	10	70	07
00200020	00	10	00	07	00	00	00	00	00	00	00	00	00	00	00	00
...																
00200FF0	00	00	00	00	00	00	00	00	00	1F	E0	07	00	1F	F0	07
...																

SISTEMAS OPERATIVOS

Segundo Curso Grado en Informática. Julio 2018

APELLIDOS, Nombre:_____

1	2a	2b	2c
0.5	0.5	0.5	0.5

1. ¿Puede un sistema en tiempo real que tiene que atender a 1024 eventos periódicos ser planificable (*schedulable*)? **(SI/NO)_SI_**

   Explicación: *Para que un sistema en tiempo real que atiende a eventos periodicos sea planificable (*schedulable*) ha de verificarse que*

   $$\sum_i \frac{C_i}{T_i} \leq 0$$

   donde C_i es el tiempo de computación necesario para tratar el evento i y T_i el periodo con el que sucede el evento i. En nuestro caso, dado que son 1024 enventos, los tiempos de computación de cada evento y sus periodos habrían de verificar

   $$\sum_{i=0}^{i=1023} \frac{C_i}{T_i} \leq 0$$

2. Un sistema tiene 3 procesos, A, B y C cuyas duraciones son 3-(4)-3, 5-(6)-1-(2)-1 y 7 respectivamente (donde x-(y)-z indica una ráfaga de CPU de duración x, seguida de una ráfaga de e/s de duración y, seguida de una ráfaga de CPU de duración z). Supondremos que

   - varios procesos pueden hacer e/s concurrentemente
   - cuando en un mismo instante coincide la llegada a la cola de listos de un proceso que viene de una e/s con uno que abandona la CPU, el que abandona la CPU se coloca al final de la cola de listos después del que viene de la e/s
   - A llega en el instante T=0, y B y C llegan en el instante T=1 (B llega antes que C)

   Se presentam dos planificaciones donde la fila CPU indica que proceso está en CPU en cada instante, COLA la cola de listos y E/S que proceso (o procesos) están en e/s

Planificación 1

T																												
CPU	A	A	B	B	C	C	A	-	B	B	C	C	B	-	A	A	C	C	A	-	C	-	B	-	-	-	B
COLA		B C	C A	C A	A B	A B	B C	B C	C	C	B	B A	A C	A C	C	C	A	A	C	C B	B	B				B	
E/S								A	A	A	A			B	B	B	B	B	B					B	B		

Planificación 2

T																											
CPU	A	A	B	B	C	C	A	B	B	B	C	C	A	A	C	C	A	B	C	-	B						
COLA		B C	C A	C A	A B	A B	B C	C	C	C		A	C	C	A	A	B C	C									
E/S								A	A	A	A B	B	B	B	B	B			B	B							

(a) Calcular los tiempos de servicio, retorno y espera para cada proceso en cada una de las planificaciones.

Planificación 1

	Ráfagas	Tiempo Servicio	Tiempo Retorno	Tiempo de Espera
Proceso A	3-(4)-3	10	19-0=19	9
Proceso B	5-(6)-1-(2)-1	15	27-1=26	11
Proceso C	7	7	21-1=20	13

Planificación 2

	Ráfagas	Tiempo Servicio	Tiempo Retorno	Tiempo de Espera
Proceso A	3-(4)-3	10	17-0=17	7
Proceso B	5-(6)-1-(2)-1	15	21-1=20	5
Proceso C	7	7	19-1=18	11

(b) ¿Constituye alguna de esas planificaciones una **IMPLEMENTACION CORRECTA** de una planificación Round Robin de quantum=2 en este sistema? **(SI/NO)_NO_**

Explicación: *La implementación 1 deja la CPU ociosa cuando hay procesos disponibles en la cola de listos, por lo que no es correcta.*

La implementación 2 asigna la CPU al proceso B en el instante 9, cuando B ya ha estado en CPU su quanto de 2, y C esta listo para ejecutarse en dicho instante

(c) En al caso de haber contestado que ninguna de las anteriores planificaciones es una implementación correcta de un Round Robin de quantum=2, proponer aquí la **IMPLEMENTACION CORRECTA** del RR de quantum=2. Si se ha contestado que alguna de ellas es una implementación correcta de un Round Robin de quantum=2, planificar dicho sistema usando el algoritmo SRTF. (**IMPORTANTE:** decir que ninguna es correcta e implementar SRTF, o decir que alguna de ellas es un RR correctamente implementado y a continuación implementar RR supondrá un 0 en **TODA** la pregunta 2)

Planificación:(SRTF/RR) _RR__

T																											
CPU	A	A	B	B	C	C	A	B	B	C	C	B	A	A	C	C	A	C	B	-	-	B					
COLA		B C	C A	C A	A B	A B	B C	C	C	B	B	A C	C	C	A	A	C										
E/S								A	A	A	A		B	B	B	B	B	B		B	B						

Apellidos: ..... **Nombre:** ..... **DNI:** .....

Sistemas Operativos – Grado Ingeniería Informática UDC. Julio 2018
Sólo puede usar lápiz, bolígrafo y calculadora. Smartphones apagados y en la mochila.
Tiempo máximo para todo el examen: **2h45m**

Parte E-S

1. **(0.75 puntos)** Un fichero de texto contiene "98765432100000". 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("num.dat",O_RDONLY);
l. 5:    pread(fd,buf,4,3);
l. 6:    lseek(fd, 3, SEEK_SET);
l. 7:    read (fd,buf+5,4);
l. 8:    write(STDOUT_FILENO,buf,10);
l. 9:    exit(0);
l.10: }
```

Imprime: 6543-6543-
Justificación
$buf[0 \ldots 9]$ está inicializado con 10 caracteres '-'
En la línea 5 se leen 4 caracteres a partir de la posición 3 del fichero: $buf[0 \ldots 3] \leftarrow 6543$
En la línea 6 se modifica el offset actual en el fichero a la posición 3 (saltan los 3 primeros bytes)
En la línea 7 se leen 4 caracteres y se colocan en buf+5: $buf[5 \ldots 8] \leftarrow 6543$
En definitiva, buf contiene 6543-6543-
En la línea 8 se muestran por pantalla los 10 caracteres de buf: 6543-6543-

2. **(0.75 puntos)** Considerando las estructura de capas del software de entrada/salida: User level software (ULS), Device Independent Layer (DIL), Device Driver (DD), e Interrupt Handler(IH). Indíquese en qué capa tendrán lugar las siguientes operaciones:

   - En un `printf` convertir un número a su representación en forma de *string* para ser escrito en la pantalla: ULS
   - Despertar a un proceso que estaba esperando por una entrada proveniente del teclado, una vez que la entrada se haya completado: IH
   - Implementar el algorithmo de planificación de disco C-look: DD
   - Determinar qué bloque de disco contiene un cierto offset de un fichero: DIL
   - Determinar cuántos sectores de disco se escriben para almacenar el i-ésimo bloque de un dispositivo: DD
   - Chequear si un determinado bloque está en la caché de bloques: DIL
   - Asignar un nuevo bloque a un fichero que necesite crecer: DIL
   - Chequear si un proceso que intenta abrir un fichero tiene los permisos necesarios DIL
   - Chequear si un descriptor de fichero es válido para lectura: DIL