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

Sistemas Operativos – Grado Ingeniera Informática UDC. Junio 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: 0.5, P2: 0.3, P3: 0.3, P4: 0.4, P5: 0.5

P1) Un sistema de archivos tipo UNIX System V tiene un tamaño de bloque de 4Kbytes, i-nodos con 12 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 8 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: 2Mbytes + 48Kbytes

Tamaño máximo usando puntero de indirección doble: 1Gbyte + 2Mbytes + 48Kbytes

Tamaño máximo usando puntero de indirección triple: 512Gbytes + 1Gbyte + 2Mbytes + 48Kbytes

P2) En ese sistema de archivos, un proceso abre el archivo "*p1.c*": *fd=open("/home/usr2/so/p1/p1.c", O_RDONLY)*; suponiendo que el usuario efectivo del proceso tiene permiso de lectura del fichero y los permisos pertinentes de acceso y lectura en los diferentes directorios, que las caches de datos e inodos están inicialmente vacías, que la entrada *home* está en el quinto bloque del directorio raíz y que la entrada *p1.c* está en el segundo bloque de su directorio padre, calcular cuántos accesos a disco son necesarios, como mínimo, para esa apertura.

Nº accesos mínimos: 11 = Nº accesos mínimos al área de datos: 10

+

Nº accesos mínimos a la lista de inodos: 1

¿Cuántas lecturas de inodos invocó el S.O.? (independientemente del estado de las caches): 6

P3) Tras los supuestos anteriores (P1 y P2), una vez abierto el archivo *p1.c,* de tamaño 0.5Gbytes, lo primero que realiza el proceso es *lseek(fd, 2097152, SEEK_SET)*. Calcula cuántos bloques tendría que leer ahora de disco el S.O. para la operación *c=fgetc(fd)*, suponiendo el Buffer Cache vacío (Nota: $2097152 = 2*2^{20}$, SEEK_SET referencia el desplazamiento desde el principio del fichero)..

Nº bloques que es necesario leer: 2

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

P4) Un proceso abre (sin retorno de ningún error) el fichero anterior "*p1.c*" (descriptor *fd1*, tamaño 0.5Gbytes, tamaño bloque en disco 4 Kbytes) y, posteriormente, duplica el descriptor de fichero abierto con el servicio *dup()*, al ejecutar el siguiente código:

```
main()
{
int fd1, fd2;
char buffer[4096];

fd1=open("/home/usr2/so/p1/p1.c", O_RDONLY);
lseek(fd1, 1024, SEEK_SET); /* SEEK_SET referencia desde el principio del fichero */
fd2=dup(fd1);
read (fd2, buffer, 2048); /* read 1 */
read (fd1, buffer, 2048); /* read 2 */
lseek(fd2, 512, SEEK_SET);
close(fd1);
close(fd2);
}
```

Contestar a lo siguiente.

A. ¿Qué valor se guarda en la variable *fd2* después de ejecutar el servicio *dup()*?

Valor fd2: 4

B. Indicar si es cierto o falso que, después de ejecutar el servicio *dup()*, el desplazamiento asociado al descriptor *fd2* queda en la posición 1024.

Cierto

C. Indicar si es cierto o falso que, después de ejecutar la segunda invocación a *lseek()*, el desplazamiento asociado al descriptor *fd2* queda en la posición 1536.

Falso

D. Las dos lecturas con las llamadas *read* leen la misma información del fichero (mismo rango de bytes).

Falso

E. La primera lectura (*read 1*) implica leer solamente 1 bloque por parte del SO. Además, esa lectura puede no ser necesariamente en el disco al existir *Buffer Cache*.

Cierto

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

P5) Supongamos la siguiente secuencia de comandos:

1. Se crea un hard link al fichero “p1.c” (cuyo número de inodo es 21288, tamaño 0.5 Gbytes y número de hard links=1) en su mismo directorio:

ln p1.c pnuevo.c

2. Se crea un *soft link* al fichero *“p1.c”* en su mismo directorio:

ln -s p1.c slink / el comando ls -l mostraría slink -> p1.c */*

3. Posteriormente se crean 2 *hard links* al fichero *slink*:

ln slink slink2 / crea hard link slink2 */*

ln slink2 slink3 / crea hard link slink3 */*

Contestar a lo siguiente:

A. Indicar el tamaño (Gbytes/Mbytes/bytes) del fichero *slink2*: 4 bytes

B. Indicar cuál es el número de *(hard) links* del fichero *slink2*: 3

C. Una vez creados los ficheros *slink*, *slink2* y *slink3*, al borrar el fichero *pnuevo.c* (*rm pnuevo.c*) se decrementa su número de *hard links*. ¿Se puede acceder al contenido del fichero con número de inodo 21288 a través del link simbólico *slink3?*

Sí

Misma pregunta a través del fichero *slink2*:

Sí

D. En la operación anterior (*rm pnuevo.c*) el S.O. borra los datos del fichero en el área de datos.

Falso

E. En la operación anterior (*rm pnuevo.c*) el S.O. indica como libre el inodo 21288 en la lista de inodos en disco.

Falso

Sistemas Operativos

Grado en Informática. Julio 2017

Apellidos, Nombre:_____

1a	1b	2	T
0.5	0.75	0.75	2

1. En un sistema con planificación por prioridades apropiativa, la prioridad de cada proceso está representada por un número entre 0 y 127, donde a menor número corresponde mayor prioridad. Se tienen 3 procesos: P_1 con una ráfaga de CPU de 8 ms, P_2 con dos rágas de CPU de 4ms separadas por una ráfaga de e/s de 4 ms, y P_3 con 4 ráfagas de CPU de 1ms separadas por 3 ráfagas de e/s de 2 ms. Suponemos que las prioridades de cada proceso permanecen constantes, que los instantes de llegada de P_1, P_2 y P_3 son 0,1 y 2 ms respectivamente y que varios procesos pueden estar en e/s concurrentemente.

   a1) ¿Podría determinarse, con los datos que se dan, un conjunto de valores de prioridades de P_1, P_2 y P_3 que hiciese mínimo el tiempo de espera promedio?. Si la respuesta es si, indíquese dicho conjunto en el siguiente cuadro. Si la respuesta es no, indíquese **NO** en el cuadro.

Prioridad de P_1	**2**
Prioridad de P_2	**1**
Prioridad de P_3	**0**

   a2) Razonar debidamente la respuesta. (si cree que no puede determinarse con los datos que se dan, indíquese el porqué)

   El algoritmo que produce los mejores tiempos de espera promedio es el SRTF, SRTF es un caso particular de planificación por prioridades apropiativas donde se asigna más prioridad al proceso de ráfaga mas corta. Como P_3 tiene ráfagas de CPU más cortas que P_2 y éste que P_1, con asignar a P_3 más prioridad que a P_2 y a éste más que a P_1 ya está. (nótese que el conjunto de valores no es único)

   b1) Con las priordades definidas en *a1)* planificar los procesos P_1, P_2 y P_3. (Si se ha contestado que no puede determinarse, utilizar las siguientes prioridades: 4 para P_1, 8 para P_2 y 2 para P_3)

T(ms)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	P_1	P_2	P_3	P_2	P_2	P_3	P_2	P_1	P_3	P_1	P_1	P_3	P_2	P_2	P_2
E/S				P_3	P_3		P_3	P_3 P_2	 P_2	P_3 P_2	P_3 P_2				

T(ms)	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
CPU	P_2	P_1	P_1	P_1	P_1										
E/S															

   b2) A partir de lo hecho en *b1)* calcular los tiempos (en ms) de retorno, servicio y espera de cada proceso. Calcular además los promedios del tiempo de retorno y del tiempo de espera.

Tiempo de retorno (turnaround time) es el tiempo transcurrido desde que el proceso llega hasta que termina. Tiempo de servicio

es el tiempo necesario para ejecutar el trabajo si este fuese el único proceso en el sistema, es decir , la suma de sus ráfagas de CPU y de entrada salida. Tiempo de espera es el tiempo que un proceso pasa en la cola de listos sin ejecutarse t es la diferencia entre en tiempo de retorno y el de servicio.

	P_1	P_2	P_3	Promedio
Tiempo de retorno	$20-0=20$	$16-1=15$	$12-2=10$	15
Tiempo de servicio	8	$4+4+4=12$	$1+2+1+2+1+2+1=10$	———
Tiempo de espera	12	3	0	5

2. 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","10",NULL);

}
```

Ademas del primer proceso creado para ejecutar dicho programa, ¿Cuantos procesos se crearán el los primeros 60 segundos? (*sleep(n)* deja un proceso en espera durante n segundos y suponemos que la llamada fork() utiliza 10ms en al proceso padre y nada en el hijo. Despreciamos lo que dura la llamada *exec*, lo que tarda *sleep* en poner en espera un proceso y lo que emplean las operaciones con enteros-control del bucle, asignaciones, ...). *./a.out* existe, es un ejecutable válido con permiso de ejecución y su código se corresponde a:

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

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

Processos creados en los primeros 60 segundos: 11990
Explicación: El primer *fork()* crea un proceso. Por tanto tenemos dos procesos, que despues de 10ms, llegan al bucle. Cada iteracciń del bucle crea un hijo que reemplaza su código y ya no crea mas procesos. Es decir, tenemos dos procesos a los que les quedan hasta los 60 segs, 59990ms para realizar un bucle en el que en cada iteracción de 10ms (la llamada *fork()*) se crea un proceso. En definitiva se crean: 1 proceso en el primer *fork()* +5999 procesos por el proceso original en el bucle + 5999 procesos por el proceso hijo (el del primer *fork()* en el bucle.

Apellidos:_____Nombre:_____ DNI: _____

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

Parte Memoria.

1. En los dos apartados de esta pregunta tiene que ser correcta la respuesta y correcta y suficiente la explicación

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

char** func1_Str();
char** func2_Str();

int main(void)
{
   char **ptr1 = NULL;
   char **ptr2 = NULL;

   ptr1 = func1_Str();
   printf("\n [%s] \n",*ptr1);

   ptr2 = func2_Str();
   printf("\n [%s] \n",*ptr2);

   printf("\n [%s] \n",*ptr1);

   return 0;
}

char** func1_Str()
{
   char *p = "Linux";
   return &p;
}

char** func2_Str()
{
   char *p = "Windows";
   return &p;
}
```

a) (0.2) Este código C, cuyo ejecutable no produce ningún error en tiempo de ejecución,produce una salida por pantalla por tres líneas. Escriba la salida y explique por qué se produce
Salida:
[Linux]
[Windows]
[Windows]

Explicación: Las funciones devuelven las direcciones de variables locales, y los printf imprimen los contenidos de esas direcciones. La segunda función utiliza la misma zona de memoria para su stack frame que la primera función, de forma que sobreescribe sus contenidos, por está razón en el tercer print los contenidos apuntados por ptr1 son los modificados por la segunda llamada a la función.

b) (0.2) ¿Cuantos stack frames se crean en la ejecución de ese programa?

Respuesta correcta:___3___
Explicación: Un stack frame para main y uno para cada llamada a función, independientemente de que la segunda llamada el stack frame vaya en la misma zona de memoria que la primera.

2. Considere la siguiente cadena de referencia a páginas de un proceso:
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

¿Cuántos fallos de página se producen si el proceso tiene asignadas 6 frames (páginas físicas), contando todos los fallos incluidos los que se producen al principio de la cadena que no implican reemplazo, y considerando que las frames inicialmente están vacías?. Rellena también la tabla con la asignación de páginas a frames

(0.2 puntos) LRU, número de fallos:____7

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

(0.2 puntos) FIFO, número de fallos:____10

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

(0.1 puntos) Optimo, número de fallos:____7

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

F* Se supuso que la página 4 tardará más en referenciarse que la 5 y por eso se eligió como víctima

3. (0.3 puntos) Con Gestión del Heap nos referimos a decisiones sobre:

a) qué páginas físicas deben contener el heap
b) qué páginas virtuales pasar a disco cuando se llena la memoria física.
c) la implementación de mmap() (para proyectar ficheros en memoria de usuario)
d) la implementación de malloc() y free()
e) qué variables dinámicas poner en el heap
f) qué variables estáticas poner en el heap
La respuesta tiene que ser correcta y correcta y suficiente la explicación

Respuesta correcta: d)
Explicación: La librería malloc gestiona la memoria dinámica para los programas evitando que estos tengan que hacer llamadas de más bajo nivel como brk(), además de darle funcionalidades de una librería de memoria.

4. (0.8 puntos) Considere un procesador con una arquitectura de direcciones virtuales de 32 bits y dos niveles de tablas de página. Los 6 bits más significativos determinan la entrada en la tabla de página de primer nivel (o tabla de página raíz), los 10 siguientes la entrada en la tabla de páginas de segundo nivel y los 16 menos significativos el offset en la página.

Para que las respuestas puntúen tiene que ser la respuesta correcta y el cálculo o explicación. Si no puede dar alguna respuesta indique por qué. En las preguntas SI/NO la contestación ambigua no puntua.

a) ¿Cuántas entradas tiene la TP de primer nivel?

$2^{\wedge 6} = 64$

b) ¿Cuántas entradas tiene una TP de segundo nivel?

$2^{\wedge 10} = 1024$

c) ¿Cuál es el tamaño de página virtual?

$2^{\wedge 16} = 64KB$

d) ¿Cuál es el número máximo de páginas virtuales que puede tener un proceso?

$2^{\wedge 6} \times 2^{\wedge 10} = 2^{\wedge 16}$ páginas

e) ¿Cuál es el número máximo de páginas físicas direccionables?

No puede saberse, se necesitaría saber los bit dejados en cada entrada de la TP para direccionar páginas físicas

f) ¿Puede obtenerse en esta arquitectura la dirección de la TP de primer nivel? Conteste SI/NO y explique

Si, con el contenido de un registro del procesador

g) ¿Puede una acceso a una TP de segundo nivel provocar un fallo de página? Conteste SI/NO y explique

Si, porque las TP están en memoria.

h) ¿Puede un Sistema Operativo ofrecer a los programas segmentación paginada con esta arquitectura? Conteste SI/NO y explique

Si, haciendo segmentación por software tal y como se vió que hace Linux sobre las arquitecturas de paginación en varios niveles.

INPUT/OUTPUT SCORE:

OPERATING SYSTEMS (Grado en Ing. Informática) 28/6/2017.

SURNAME, NAME: .....

1. (**1.0 puntos**) Indíquese qué sucede al ejecutar el siguiente programa. Se asume que el fichero file1.dat existe en el directorio actual y el comando cat file1.dat nos muestra que su contenido es 0123456789. Además el fichero file2.dat no existe actualmente. *(a)* ¿Qué se hace en la línea l.11?; *(b)* ¿qué se escribe en pantalla?; *(c)* ¿ qué se escribe en file2.dat? Razone sus respuestas.

```
l. 1: #include "includes.h"
l. 2 int main() {
l. 3   char buf[20];  int i=0,ifd,ofd,bk;
l. 4   ifd = open("file1.dat",O_RDONLY);
l. 5   ofd = open("file2.dat",O_WRONLY|O_CREAT, 0666);
l. 6   bk=dup(STDOUT_FILENO);
l. 7   close(STDOUT_FILENO);
l. 8   close(STDIN_FILENO);
l. 9   dup(ifd);
l.10   dup(ofd);
l.11   while (read(STDIN_FILENO,buf+i,1)) i++;
l.12   write(STDOUT_FILENO,buf,i);
l.13   close(STDOUT_FILENO);
l.14   dup(bk);
l.15   write(STDOUT_FILENO,"done\n",5);
l.16   close(bk); close(ifd); close(ofd);
l.17 } //read() devuelve 0 cuando no hay m\'as datos que leer
```

   (a) Asumiendo que el contenido de la tabla de ficheros abiertos del proceso al ejecutar las líneas l.2 y l.5 fuese el que indica a continuación, indíquese cuál es el contenido de las entradas 0 a 5 de dicha tabla, en la línea l.11.

	estado l.2
0	teclado-in
1	pantalla-out
2	pantalla-err
3	
4	
5	
6	

	estado l.5
0	teclado-in
1	pantalla-out
2	pantalla-err
3	file1.dat
4	file2.dat
5	
6	

	estado l.11
0	file1.dat
1	file2.dat
2	pantalla-err
3	file1.dat
4	file2.dat
5	pantalla-out
6	

(b) ¿Qué se hace en la línea l.11? Razone su respuesta (¿Se leen/escriben datos? los introduce el usuario por teclado? ¿Cuántos? ¿Cuáles? Indique el valor de i y el contenido de buf tras ejecutar la línea l.11).

en l.7 y l.8 se cierran la salida y la entrada estándar.
en l.9 se redirecciona la entrada estándar a file1.dat (abierto en l.4 en modo lectura);
en l.10 se redirecciona la salida estándar a file2.dat. De este modo:

En l.11 se leen todos los datos del fichero file1.dat (0123456789\n) y se colocan en buf.
i indica núm de bytes leídos (11 incluyendo el \n; o 10 si no nos damos cuenta del \n)

i vale: 11 (o 10 si obviamos el \n) ; buf contiene: 0123456789\n

(c) ¿qué se escribe en file2.dat? Razone su respuesta.

En l.12 se escriben en el fichero file2.dat, los i bytes que hay en buf[0..i-1]

(d) ¿qué se escribe en pantalla? Razone su respuesta.

En l.14 y l.15 se deshace la redirección en la salida estándar, así que el write de l.15 permite esribir 'done' en pantalla.

2. (**0.5 puntos**) Indíquese con qué método(s) de entrada/salida (polling, E/S por interrupciones o E/S por DMA) se relacionan las siguientes sentencias:

- La CPU debe suministrar al controlador el tipo de operación (lectura o escritura), la dirección de transferencia de datos y cantidad de bytes a transferir.

  E/S por DMA

- Durante la transferencia de varios datos, la sincronización se consigue al preguntar la CPU al dispositivo si está listo.

  polling