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

Sistemas Operativos – Grado Ingeniería Informática UDC. Enero 2020
Sólo puede usar lápiz, bolígrafo y calculadora.
Tiempo máximo para todo el examen: 2h45m

Parte Sistema Ficheros
(Cada apartado puntúa de modo proporcional a la puntuación total de la pregunta. Cada contestación en blanco resta la mitad de esa puntuación proporcional, cada contestación incorrecta resta esa puntuación proporcional).
Puntuación preguntas: P**1**: 0.7, P2: 0.4, P3: 0.9

P1) Un sistema de archivos tipo UNIX System V tiene un tamaño de bloque de 2Kbytes, i-nodos con 10 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 8 bytes. Calcular cuántos bloques son necesarios en el área de datos para representar un fichero con un tamaño de 130 Mbytes+1byte, diferenciando entre bloques de datos y bloques de índices.

Nº bloques de datos: 65Kbloques+1 = 66561

Nº de bloques de índices: 263

Fragmentación interna del fichero: 2047 bytes

P2) En ese sistema de archivos UNIX (tamaño de bloque de 2Kbytes), el *boot* ocupa los 3 primeros bloques de su partición de disco. Por tanto, el superbloque comienza en el bloque lógico 3 de la partición de disco del SF (se numera a partir del bloque 0). El fichero “datos” tiene asociado el *inodo* 643 y ocupa el bloque (lógico) 33 desde el principio de la partición. El tamaño del *inodo* es de 64 bytes. Calcula el tamaño (en Kbytes, no bloques) del *superbloque* y el bloque lógico (desde el principio de la partición) correspondiente al inodo 3201.

Tamaño del superbloque: 20 Kbytes

Bloque del inodo 3201: 113

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

P3) En el sistema de archivos del problema P1 (tamaño bloque 2Kbytes, 10 punteros directos, …), tenemos el archivo *"/home/juan/so/p1/p1.c"* con un tamaño de 3Mbytes e inodo número 6549. Al ejecutar un proceso (con el código principal indicado a continuación), el usuario efectivo del proceso coincide con el propietario del fichero (*p1.c*), y tiene además los permisos de acceso y lectura a los directorios *home*, *juan*, *so* y *p1*. Las cachés de datos e inodos están inicialmente vacías y la entrada *p1* está en el octavo bloque de su directorio padre (*so*), mientras los demás entradas están en el primer bloque de su directorio padre. Indicar lo siguiente referente al trozo de código:

```
struct stat buf, char c;
chmod("/home/juan/so/p1/p1.c", 0420);
if (lstat ("/home/juan/so/p1/p1.c", &buf)!=-1){
   printf("%s", convertir_permisos(buf.st_mode));     /* se convierten los permisos a
                                                        formato rwxrwxrwx y se imprimen*/
    int fd1=open("/home/juan/so/p1/p1.c", O_RDONLY);     /* es la primera apertura
                                                            de fichero del proceso */
   link("p1.c", "practica1.c");          /* se crea hard link practica1.c */
   symlink("practica1.c", "slink_practica1.c" );        /* se crea link simbólico a practica1.c */
   int fd2=open("/home/juan/so/p1/practica1.c", O_RDONLY);        /* segunda apertura */
   lseek(fd2, 2097152, SEEK_SET);  /* SEEK_SET indica que el desplazamiento
                                      se considera a partir del origen del fichero (2097152 = 2^21)  */
    c=fgetc(fd2);
   close(fd2); close(fd1);
   unlink("p1.c");    }
```

A. ¿Cuál es el número de accesos necesarios a disco, únicamente en el área de datos, en la primera apertura del fichero p1.c?: 12

B. ¿Cuál es el valor asignado al descriptor de fichero *fd2*?: 4

C. Indica el número de bloques que el S.O. necesita leer en disco para obtener el valor de *c* en *fgetc(fd2*): 3

D. ¿Cuál es el tamaño del fichero *"slink_practica1.c": 11 bytes* *(indicar unidad: bytes, Kbytes , …)*

E. *fgetc* es una llamada al sistema operativo. Cierto/Falso: F

F. *link* es una llamada al sistema operativo. Cierto/Falso: C

G. *unlink* va a eliminar la entrada de directorio "p1.c" en su directorio padre pero no libera el inodo 6549. Cierto/Falso: C

H. El lincador encuentra el código de *printf* en la librería estándar de C (versión estática o dinámica de la librería). Cierto/Falso: C

I. Indica *los* permisos del fichero *p1.c* impresos en formato *rwxrwxrwx*: r-- -w- ---

Apellidos:_____Nombre:_____ DNI: _____

Sistemas Operativos - Grado Ingeniera Informática UDC. Enero 2020
Sólo puede usar lápiz, bolígrafo y calculadora.
Tiempo máximo para todo el examen: **2h45m**

Parte Memoria.

1. (1 puntos) Responda Verdadero/Falso (o no conteste) a cada pregunta (0.05 cada una) sin justificaciones, pero cada respuesta incorrecta anula una correcta. La puntuación mínima es zero para esta pregunta.

-En la pila del proceso están las variables locales y parámetros de funciones: ____V_____
-Las variables locales del main no están en la pila del proceso: ___F_____
-Las variables automáticas definidas en funciones están en la pila del proceso: __V__
-Las variables automáticas definidas en main() no están en la pila del proceso:___F___
-El código y los datos de la librería malloc usada en un programa C están en el espacio de direcciones del kernel:__F___

-Las variables estáticas definidas en funciones están en la pila del proceso: ___F___
-Las variables estáticas definidas en main() no están en la pila del proceso: ___V___
-Para un proceso con varios hilos de ejecución, cada hilo tiene su propio espacio de direcciones (código, datos, stack): ___F____
-Para un sistema con una tabla de páginas en un nivel (y no caché, no TLB), traer y ejecutar una instrucción que añade un valor constante a un registro, implica exactamente dos accesos a memoria: ___V___
-Para un sistema con una tabla de páginas de dos niveles (y no caché, no TLB), traer y ejecutar una instrucción que añade un valor constante a un registro, implica exactamente dos accesos a memoria: ___F___

-Para un sistema con tablas de páginas en dos niveles donde la página raíz no está fija en memoria y procesador con caché y TLB, traer y ejecutar una instrucción que añade un valor constante a un registro, puede implicar ningún acceso a memoria: ___V___
-Dado un número de bits para las direcciones virtuales, y una tabla de páginas de un nivel, el tamaño de la tabla de páginas se reduce con páginas más grandes: ____V____
-Si las direcciones físicas son de 32 bits y las páginas de 4Kbytes, el número de página física viene dado por 18 bits:____F_____
-Una tabla de páginas multinivel típicamente reduce la cantidad de memoria necesaria para almacenar tablas de páginas comparada con una tabla de páginas de un nivel:____V____
-Si el bit de validez está a cero en una Entrada de Tabla de Página necesaria para un acceso a memoria, la página deseada deberá ser traída a memoria desde el dispositivo de almacenamiento:___F___

-Con Tablas de Páginas Invertida, las páginas virtuales pueden ser más grandes que las páginas físicas:____F____
-Las TLBs son más beneficiosas con tablas de páginas multinivel que con tablas de páginas de un nivel:____V____
-Cuando el dirty bit está a cero en una Entrada de Tabla de Página necesaria para un acceso a memoria, existe una copia exacta de la página deseada en el dispositivo de almacenamiento:____V____

-El algoritmo de reemplazo LRU con N+1 páginas de memoria siempre se comporta mejor que LRU con N páginas de memoria:____F_____
-El algoritmo de reemplazo FIFO con N+1 páginas de memoria siempre se comporta mejor que FIFO con N páginas de memoria:_____F____

2. a) (0.50 puntos) Considera un sistema con un direccionamiento de memoria física de 8GBytes, tamaño de página de 8Kbytes y tamaño de una entrada de tabla de páginas de 4 bytes. ¿Cuántos niveles de tablas de páginas son necesarios para manejar direcciones virtuales de 46 bits si cada tabla páginas es de tamaño una página? Debe indicar los cálculos para que la respuesta puntúe.
Número de niveles:__**3** _

tamaño de página 8Kbytes = 2^{13} bytes.
2^{13} bytes / 2^{2} bytes por entrada de tabla páginas = 2^{11} entradas por tabla de páginas.

Una tabla de páginas puede direccionar 2^{11} entradas x 2^{13} bytes = 2^{24} bytes
Con dos niveles de tablas de páginas se puede direccionar 2^{11} entradas x 2^{11} entradas x 2^{13} bytes = 2^{35} bytes
Con tres niveles de tablas de páginas se pueden direccionar 2^{11} entradas x 2^{11} entradas x 2^{11} entradas x 2^{13} bytes = 2^{46} bytes

b) (0.25 puntos) Para la solución anterior, ¿de cuantos bits de control (bit referencia, bit R/W, dirty bit, etc) y bits no usados (se pide la suma total de ambos tipos), se dispone en una entrada de tabla de página de cada nivel? Debe indicar los cálculos para que la respuesta puntúe.

Memoria física son 8Gbytes= 2^{33} bytes, y las páginas son de 8Kbytes = 2^{13} bytes. Por tanto hay $2^{33}/2^{13} = 2^{20}$ páginas físicas, y se necesitan 20 bits en la entrada de tabla de páginas para direccionar una páginas física, y ya que las PTE son de 32 bits, quedan por tanto 12 bits para bits de control y no usados. Eso es lo mismo para todas los niveles.

c) (0.25 puntos) Imagine que con la solución anterior el procesador dispone de caché de datos e instrucciones y TLB, en este caso, para una operación de lectura de un byte de memoria (debe dar la explicación correcta para que la respuesta puntúe),
¿cuál sería el número mínimo de accesos a memoria? : ___**0**____
el mapping de dirección virtual a dirección física se encuentra en la TLB y el byte de memoria en la caché.
¿cuál sería el número máximo de accesos a memoria?:____**4**____
3 accesos a memoria para cada nivel de TP más el acceso a memoria para conseguir el byte de memoria que no está en el caché

SISTEMAS OPERATIVOS

Segundo curso. Grado en Informática. ENERO 2020

APELLIDOS (mayúsculas), Nombre:_____

Tiempo: 2h 45m

1	2	T
1	1	2

1. Sea el siguiente código en C (con todos los *includes* necesarios y que compila correctamente), que produce un ejecutable llamado `a.out` (*sleep(n)* deja al proceso en espera durante n segundos)

```
#define VECES 32
main(int argc, char * argv[])
{
  int i;
  pid_t pid;

  for (i=0; i<VECES; i++){
      pid=fork();
      if (pid==-1)
         break;
      printf ("/%ld/",(long) pid);
  }
  sleep (60);
}
```

La salida producida por dicho código es

a) imprime '/0/' 32 veces

b) imprime '/un número positivo/' 32 veces (no necesariamente el mismo número las 32 veces)

c) imprime '/0/' y '/un número positivo/' 32 veces en total

d) imprime '/0/' y '/un número positivo/' 32 veces cada uno

e) imprime '/0/' 2^{32} veces

f) imprime '//un número positivo/' 2^{32} veces

g) imprime '/0/' y '/un número positivo/' 2^{32} veces en total

h) imprime '/0/' y '/un número positivo/' 2^{32} cada uno

i) imprime '/0/' 32! veces

j) imprime '/un número positivo/' 32! veces

k) imprime '/0/' y '/un número positivo/' 32! veces en total

l) imprime '/0/' y '/un número positivo/' 32! veces cada uno

m) imprime '/0/' tantas veces como pueda durante 60 segundos

n) imprime '/un número positivo/' tantas veces como pueda durante 60 segundos

o) imprime '/0/' y '/un número positivo/' tantas veces como pueda durante 60 segundos

p) imprime '/0/' infinitas veces

q) imprime '/un número positivo/' infinitas veces

r) imprime '/0/' y '/un número positivo/' infinitas veces

s) no imprime nada

t) imprime '/0/' un número finito de veces, menor que 2^{32}

u) imprime '/un número positivo/' un número finito de veces, menor que 2^{32}

v) imprime '/0/' y '/un número positivo/' un número finito de veces, menor que 2^{32}

w) ninguna de las anteriores

La respuesta correcta es: v,w

EXPLICACION:

w) A la primera iteración (i=0) llega un proceso, la llamada *fork()* crea un proceso y por tanto 2 procesos pasan por el `printf` imprimeendo uno de ellos /0/ (el proceso hijo) y el otro el pid del hijo (*/un número positivo/*). A la segunda iteración (i=1) llegan dos procesos, que crean otros dos, imprimiendo 2 veces /0/ y dos veces */un número positivo/*. En general en la iteración i, hay 2^i procesos que imprimen /0/ y 2^i procesos que imprimen */un número positivo/*, con lo cual, al final de bucle se imprimirá /0/ $2^0 + 2^1 + 2^2 + .. + 2^{31}$ veces y otras tantas veces */un número positivo/*. Es decir, se imprimirá $2^{32} - 1$ veces cada cosa.

v) En la última iteración del bucle tendríamos 2^{31} procesos que imprimen /0/ y 2^{31} procesos que imprimen */un número positivo/*, en total 2^{32} procesos. Es sumamente improbable que el S.O. tenga una tabla de procesos capaz de alojar 2^{32} procesos (sin contar el resto de procesos que haya en el sistema en ese momento) por lo tanto, cuando la tabla de procesos esté llena (no se pueden crear más procesos) la llamada *fork()* empezar a devolver -1, produciendo la salida del bucle y por tanto, haciendo que por este motivo (no se pueden crear tantos procesos) la respuesta `v` es válida siempre que se haya razonado adecuadamente

2. Un sistema monoprocesador con múltiples colas tiene tres colas: SY para procesos del sistema, IT para procesos de usuario interactivos y NI para procesos de usuario no interactivos. La planificación entre las colas es por prioridades apropiativas (un proceso de una cola solo podrá ejecutarse si no hay procesos listos en las colas de mayor prioridad) siendo la prioridad más alta para la cola de procesos del sistema y la más baja para los procesos de usuario no interactivos. En el cuadro siguiente se muestran los procesos del sistema en un instante dado (x-(y)-z representa una ráfaga de CPU de duración x, seguida una de e/s de duración y seguida de otra de CPU de duración z). Suponemos que el sistema usa Round Robin de *quantum* 3 para los procesos del sistema, Round Robin de *quantum* 1 para los interactivos y que los no interactivos se planifican con SJF

	Ráfagas	Tiempo de llegada	Tipo
Proceso A	4-(7)-2	0	sistema
Proceso B	2-(7)-4	1	sistema
Proceso C	2-(3)-2-(3)-1	0	interactivo
Proceso D	1-(3)-1-(3)-1	2	interactivo
Proceso E	4	0	no interactivo
Proceso F	3	2	no interactivo

Rellenar el siguiente cuadro con la planificación

CPU	**A**	**A**	**A**	**B**	**B**	**A**	**C**	**D**	**C**	**F**	**F**	**D**	**B**	**B**	**B**	**A**	**A**	**B**	**C**	**D**	**C**	**F**	**E**	**E**	**C**	**E**	**E**
SY		**B**	**B**	**A**	**A**									**A**	**A**	**B**	**B**										
IT	**C**	**C**	**C** **D**	**C** **D**	**C** **D**	**C** **D**	**D**	**C**					**C**	**C**	**C**	**C** **D**	**C** **D**	**C** **D**	**D**	**C**							
NI	**E**	**E**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E**	**E**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E** **F**	**E**			**E**		
E/S						**B**	**B** **A**	**B** **A**	**B** **A** **D**	**B** **A** **D** **C**	**B** **A** **D** **C**	**B** **A** **C**	**A** **D**	**D**	**D**							**C**	**C**	**C**			

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

Sistemas Operativos – Grado Ingeniería Informática UDC. Enero 2020 **A**

Sólo puede usar lápiz, bolígrafo y calculadora.

Tiempo máximo para todo el examen: **2h45m**

Parte E-S [1.50 puntos]

1. (**0.50 puntos**) Disponemos de un disco duro con 2000 cilindros numerados del 0 al 1999. La cabeza está situada en el cilindro 605 atendiendo una petición de acceso al cilindro 605 justo después de haber atendido la petición anterior en el cilindro 602.

   Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: $\langle 400, 300, 1550, 900, 201, 495\rangle$, indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

SSTF:	495	400	300	201	900	1550
SCAN:	900	1550	495	400	300	201
C-LOOK:	900	1550	201	300	400	495

   Nota, asúmase que no llegarán más peticiones mientras se atienden las peticiones actuales.

2. (**0.50 puntos**) Indique V/F (Verdadero/Falso) en las siguientes sentencias:

   - [F] El registro de datos de un controlador de dispositivo permite enviar comandos para indicar al dispositivo qué operación debe realizar.
   - [F] El *manejador de interrupciones* es la capa del software de Entrada/Salida que chequea si un determinado bloque está en la caché de bloques.
   - [F] Un controlador DMA configurado en *modo bus transparente* (*Transparent bus DMA*) tiene mayor prioridad de acceso al bus que si usase *modo por robo de ciclos* (*Cycle-stealing DMA*).
   - [V] Los *major numbers* de los dispositivos de disco `/dev/sda` y `/dev/sda1` son idénticos.
   - [F] Determinar qué bloque de disco contiene un cierto offset de un fichero se realiza en la capa del subsistema de E/S denominada *device driver*.

3. (**0.25 puntos**) Disponemos de un disco con un plato y dos caras por plato que contiene 2048 cilindros, y 512 sectores por pista. Cada sector contiene 512bytes.

   ¿cuántos bloques ve el sistema operativo si lo formateamos con bloques de 2048 bytes?

   Ponga una "X" en el recuadro correspondiente y justifique su respuesta.

   [ ] 1024; [ ] 2048; [ ] 4096; [X] Otro (indique abajo cuál)

   Justifique brevemente su respuesta:
   Tengamos en cuenta que puesto que tenemos un disco con dos caras, cada cilindro tiene 2 pistas. Entonces:
   $2048 \cancel{cil} \times \frac{2\ pista}{\cancel{cil}} \times \frac{512\ sect}{\cancel{pista}} \times \frac{1\ bloques}{4\ \cancel{sect}} = 2^{11} \times 2 \times 2^{9} \times 2^{-2}$ bloques $= 2^{19}$ bloques.

4. (**0.25 puntos**) El fichero "fichero.txt" contiene: `"01234567890123456789\n"`, Indique cuál será el contenido de BUF[i] ($i \in \{0..,4\}$) TRAS ejecutar la intrucción en la línea "09"del siguiente programa:

Tras l.09: BUF[i]=	3	4	-	-	\0
posición i=	0	1	2	3	4

```
l.01.  #include "includes.h"
l.02.  int main(){
l.03.    char BUF[5]= {'-','-','-','-','\0');
l.04.    int fd = open("fichero.txt",O_RDONLY);
l.05.    int fd2 = dup(fd);
l.06.    lseek(fd2, 2, SEEK_SET);
l.07.    pread(fd,BUF,1,0);        //leo "0", BUFF contine 0,-,-,-,\0  (lseek no afecta a pread)
l.08.    lseek(fd, 2, SEEK_CUR);
l.09.    pread(fd,BUF,2,3);        //leo "34", BUFF contine 3,4,-,-,\0
l.10.    read(fd,BUF,1);
l.11.    read(fd,BUF+3,2);
l.12.    printf("%s",BUF); close(fd);
l.13.  }  //Nota: sabemos que al ejecutar dicho código no se produjo ningún error de ejecución.
```