Apellidos:_____ Nombre:_____ DNI: _____

Sistemas Operativos – Grado Ingeniería Informática UDC. Junio 2023
Sólo puede usar lápiz, bolígrafo y calculadora.

Parte **Sistema Ficheros (**2 puntos)

Examen sin identificación completa NO se califica.
Esta parte NO admite entrega de hojas adicionales. Hay que contestar en las páginas y espacios (recuadros) proporcionados.

Puntuación preguntas: P1: 0.6, P2: 0.2, P3: 0.6, P4: 0.6

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 4 bytes. Calcular cuántos bloques son necesarios en el área de datos para representar un fichero con un tamaño de 516 Mbytes+1048 bytes, diferenciando entre bloques de datos y bloques de índices.

Un sistema de ficheiros tipo UNIX System V ten un tamaño de bloque de 2Kbytes, i-nodos con 10 enderezos directos, un indirecto simple, un indirecto dobre e un indirecto triple. Usa enderezos de bloque de 4 bytes. Calcula cantos bloques son necesarios na área de datos para representar un ficheiro cun tamaño de 516 Mbytes+1048 bytes, diferenciando entre bloques de datos e bloques de índices.

A UNIX file system, System V type, has a block size of 2Kbytes, i-nodes with 10 direct addresses, a single indirect, a double indirect and a triple indirect address. It uses 4-byte block addresses. Calculate how many blocks are needed in the data area to represent a file with a size of 516 Mbytes+1048 bytes, differentiating between data blocks and index blocks.

- Nº bloques de datos / Nº bloques de datos / Number of data blocks: **258K+1** (0.30 p)
- Nº de bloques de índices / Nº de bloques de índices / Number of index blocks: **519** (0.25p)
- Fragmentación interna del fichero/
  Fragmentación interna do ficheiro/ Internal file fragmentation: **1000** bytes (0.05p)

Apellidos:_____ Nombre:_____ DNI: _____

P2) En ese sistema de archivos UNIX (P1, 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 60 (los inodos comienzan con el inodo 1) y ese inodo está en el bloque (lógico) 13 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.

Nese sistema de ficheiros UNIX (P1, tamaño de bloque de 2Kbyte), o *boot* ocupa os 3 primeiros bloques da partición do disco. Polo tanto, o superbloque comeza no bloque lóxico 3 da partición do disco do SF (númerase dende o bloque 0). O ficheiro "datos" ten o inodo 60 asociado (os inodos comezan co inodo 1) e ese inodo está no bloque (lóxico) 13 desde o inicio da partición. O tamaño do inodo é de 64 bytes. Calcula o tamaño (en Kbytes, non en bloques) do superbloque e o bloque lóxico (desde o inicio da partición) correspondente ao inodo 3201.

On that UNIX file system (P1, 2Kbyte block size), *boot* occupies the first 3 blocks of its disk partition. Therefore, the superblock starts at logical block 3 of the disk partition of the SF (it is numbered from block 0). The file "data" has inode 60 associated with it (inodes start with inode 1) and that inode is in (logical) block 13 from the beginning of the partition. The inode size is 64 bytes. Calculate the size (in Kbytes, not blocks) of the superblock and the logical block (from the beginning of the partition) corresponding to inode 3201.

Tamaño del superbloque/Tamaño do superbloque/Size of superblock: **18** Kbytes

Bloque del inodo 3201/ Bloque do inodo 3201/Block of inode 3201: **112**

P3) En el sistema de archivos del problema P1 (tamaño bloque 2Kbytes, 10 punteros directos, …), tenemos el archivo *"/home/juan/so/p1.c"* con un tamaño de 3Mbytes e inodo número 6549 con número de hard links inicial de 2. 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 *raíz*, *home*, *juan*, y *so*. Las cachés de datos e inodos están inicialmente vacías y la entrada *p1.c* está en el segundo 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:

No sistema de ficheiros do problema P1 (tamaño de bloque 2Kbytes, 10 punteiros directos,...), temos o ficheiro *"/home/juan/so/p1.c"* cun tamaño de 3Mbytes e inodo número 6549 con número de hard links inicial de 2. Cando ao executar un proceso (co código principal que se indica a continuación), o usuario efectivo do proceso coincide co propietario do ficheiro (*p1.c*), e tamén ten permisos de acceso e lectura aos directorios *raíz*, *home*, *juan*, e *so*. Os cachés de datos e inodos están inicialmente baleiros e a entrada *p1.c* está no segundo bloque do seu directorio pai (*so*), mentres que as outras entradas están no primeiro bloque do seu directorio pai. Indica o seguinte sobre a peza de código:

In the file system of problem P1 (block size 2Kbytes, 10 direct pointers, ...), we have the file *"/home/juan/so/p1.c"* with a size of 3Mbytes and inode number 6549 with number of initial hard links 2. When executing a process (with the main code indicated below), the effective user of the process coincides with the owner of the file (*p1.c*), and also has access and read permissions to the directories *root*, *home*, *juan*, and *so*. The data and inode caches are initially empty and the entry *p1.c* is in the second block of its parent directory (*so*), while the other entries are in the first block of their parent directory. Indicate the following regarding the piece of code:

Apellidos:_____ Nombre:_____ DNI: _____

```
struct stat buf, char c;
int fd1,fd2,fd3;
chmod("/home/juan/so/p1.c", 0752);
printf("%s", convertir_permisos(0752)); /* se convierten los permisos en octal a formato rwxrwxrwx y se imprimen*/
                                        /* se converten os permisos en octal a formato rwxrwxrwx e se imprimen*/
                                        /* permissions in octal are converted to format rwxrwxrwx and printed*/
fd1=open("/home/juan/so/p1.c", O_RDONLY); /* es la primera apertura de fichero del proceso */
                                          /* é a primeira apertura de ficheiro do proceso */
                                          /* it is the first file open of the process*/
link("/home/juan/so/p1.c", "/home/juan/so/practica1.c"); /* se crea hard link practica1.c */
                                                         /* créase hard link practica1.c */
                                                         /*hard link practica1.c is created*/
symlink("/home/juan/so/practica1.c", "/home/juan/so/slink_practica1.c");
                                                         /* se crea link simbólico a practica1.c */
                                                         /* créase link simbólico a practica1.c */
                                                         /* soft link to practica1.c is created*/
fd2=open("/home/juan/so/practica1.c", O_RDONLY);  /*segunda apertura */ /* second opening */
lseek(fd2, 524288, SEEK_SET); /*524288 = 2^19*/
                              /* SEEK_SET indica que el desplazamiento se considera a partir del origen del fichero */
                              /* SEEK_SET indica que o desprazamiento considérase desde a orixe do ficheiro */
                              /* SEEK_SET specifies that the offset is considered from the origin of the file */
c=fgetc(fd2);
fd3=dup(fd2);
close(fd2); close(fd1); close (fd3);
unlink("/home/juan/so/p1.c");
```

A. ¿Cuál es el número de accesos necesarios a disco, únicamente en el área de datos, en la primera apertura de *p1.c*?:
Cal é o número de accesos ao disco necesarios, só na área de datos, ao abrir *p1.c* a primeira vez?:
What is the number of disk accesses required, only in the data area, on the first opening of *p1.c*?: **5**

B. Indica el número de bloques que el SO necesita leer en disco para obtener el valor de *c* en *fgetc(fd2)*:
Indica o número de bloques que o SO cómpre ler desde o disco para obter o valor de *c* en *fgetc (fd2)*:
Indicate the number of blocks that the OS requires to read from disk to get the value of *c* in *fgetc (fd2)*: **2**

C. ¿Cuál es el valor asignado al descriptor de fichero fd3?:
Cal é o valor asignado ao descriptor de ficheiro *fd3*?:
What is the value assigned to the file descriptor *fd3*?: **5**

D. Al ejecutar *unlink*, ¿cuántos bloques de datos (no índices) del fichero *p1.c* se liberan en el área de datos?:
Ao executar *unlink*, cantos bloques de datos (non índices) do ficheiro p1.c se liberan na área de datos?:
When *unlink* is run, how many data blocks (not indexes) of file *p1.c* are released in the data area?: **0**

E. ¿Cuál es el número de hard links de slink_practica1.c después de ejecutar *unlink*?:
Cal é o número de *hard links* de *slink_practica1.c* despois de executar *unlink*?:
What is the number of *hard links* of *slink_ practica1.c* after running *unlink*: **1**

F. Indica los permisos del fichero que se imprimen en formato *rwxrwxrwx*:
Indica os permisos de ficheiro que se imprimen en formato *rwxrwxrwx*:
Indicate the file permissions that are printed in *rwxrwxrwx* format: **rwx r-x -w-**

Apellidos:_____ Nombre:_____ DNI: _____

P4) Indicar si es cierto/falso en cada pregunta.

Cada apartado puntúa 0.1p. Cada respuesta errónea puntúa -0.1. Cuestiones no respondidas no puntúan. La puntuación mínima de P4 es 0, es decir, en ningún caso P4 lleva a puntuación negativa para el total del examen.

A. El S.O. puede abrir varias veces un fichero y con distintos modos de apertura, y ese número de aperturas se mantiene en el inodo en memoria, pero no en el inodo en la Lista de Inodos en disco. **C**

B. Al lincar un código con una librería dinámica, las funciones necesarias de la librería se integran en el fichero ejecutable. **F**

C. Al ejecutar la función *printf* de C se incrementa tanto el tiempo de ejecución en modo usuario como en modo sistema. **C**

D. Es posible crear *hard links* entre diferentes sistemas de ficheros montados. **F**

E. La idea fundamental de un sistema de ficheros Unix basado en registro (*journaling file system*) es llevar control/registro de las creaciones y eliminaciones de los ficheros a lo largo del tiempo. **F**

F. El *Buffer Cache* reduce tanto el número de lecturas como de escrituras físicas sobre los discos montados. **C**

Indique se é verdadeiro/falso en cada pregunta.

Cada apartado puntúa 0.1p. Cada resposta incorrecta puntúa -0.1p. Cuestións non respondidas non puntúan. A puntuación mínima para P4 é 0, é dicir, en ningún caso P4 ten puntuación negativa para o exame total.

A. O S.O. pode abrir un ficheiro varias veces e con diferentes modos de apertura, e ese número de aperturas gárdase no inodo da memoria, pero non no inodo na Lista de inodos do disco. ____

B. Ao lincar un código cunha librería dinámica, as funcións necesarias da librería se integran no ficheiro executable. ____

C. Ao executar a función *printf* de C se incrementa tanto o tempo de execución en modo usuario como en modo sistema. ____

D. É posible crear *hard links* entre diferentes sistemas de ficheiros montados. ____

E. A idea fundamental dun sistema de ficheiros Unix basado en rexistro (*journaling file system*) é levar un control/rexistro das creacións e eliminacións dos ficheiros ao longo do tempo. ____

F. O *Buffer Cache* reduce tanto o número de lecturas como de escrituras físicas nos discos montados. ____

Indicate whether it is true/false for each question.

Each question scores 0.1p. Each wrong answer scores -0.1p. Unanswered questions do not score. The minimum score for P4 is 0, that is, in no case does P4 have a negative score for the total exam.

A. The O.S. can open a file several times and with different opening modes, and that number of openings is kept in the inode in memory, but not in the inode in the Inode List on disk. ____

B. When linking a code with a dynamic library, the necessary functions of the library are integrated into the executable file. ____

C. Executing the C *printf* function increases both the runtime in user mode and in system mode. ____

D. It is possible to create *hard links* between different mounted filesystems. ____

E. The fundamental idea of a *Unix journaling file system* is to keep track/record creations and deletions of files over time. ____

F. *Buffer Cache* reduces both the number of physical reads and writes on mounted disks. ____

Apellidos:_____Nombre:_____ DNI: _____

Examen sin identificación completa NON se califica
Sistemas Operativos - GEI UDC. Xullo 2023. Memoria (2 puntos)
Esta parte NON admite entrega follas adicionais. Hay que contestar nas páxinas e espacios proporcionados.

1. (1 punto) **Cada una de las preguntas siguientes son de responder V/F (Verdadero/Falso) o no responder. Respuesta correcta: +0.1; Respuesta incorrecta: -0.1. (0.1 x 10 = 1 punto) Puntuación mínima para esta pregunta: 0.**
Cada unha das preguntas seguintes son de responder V/F (Verdadeiro/Falso) ou non responder. Resposta correcta: +0.1; Resposta incorrecta: -0.1. (0.1 x 10 = 1 punto) Puntuación mínima para esta pregunta: 0.
Answer T/F (True/False) or leave the answer blank. Correct answer: 0.1; Wrong answer: -0.1 (0.1 x 10 = 1 point) Minimum score for this question: 0.

1	2	3	4	5	6	7	8	9	10
F	F	F	V	V	V	F	F	V	V

1. **En sistemas con tablas de páginas multinivel, sólo las paginas del último nivel del la tabla de páginas almacenan la dirección de páginas físicas.**
Nos sistemas con táboas de páxinas de varios niveis, só as páxinas do última nivel da taboa de páxinas almacenan o enderezo de páxina físicas.
On systems with multilevel page tables, only the pages at the last level of the page table store the address of physical pages.
2. **Las entradas de una tabla de páginas invertida, además de bits de control y otros usos, almacenan la dirección de la página física que se corresponde con una página lógica.**
As entradas dunha táboa de páxinas invertidas, ademais dos bits de control e outros usos, almacenan o enderezo da páxina física que corresponde a unha páxina lóxica.
The entries of an inverted page table, in addition to control bits and other uses, store the address of the physical page that corresponds to a logical page.
3. **Una llamada al sistema fork() con el mecanismo copy-on-write no copia la tabla de páginas del proceso padre en el proceso hijo.**
Unha chamada ao sistema fork() co mecanismo de copy-on-write non copia a táboa de páxinas do proceso pai no proceso fillo.
A fork() system call with the copy-on-write mechanism does not copy the page table from the parent process to the child process.
4. **Una llamada al sistema vfork() no copia la tabla de páginas del proceso padre en el proceso hijo.**
Unha chamada ao sistema vfork() non copia a táboa de páxinas do proceso pai no proceso fillo.
A vfork() system call does not copy the page table from the parent process to the child process.
5. **Un servicio de fallo de página siempre tiene que actualizar la tabla de páginas del proceso.**
Un servizo de erros de páxina sempre ten que actualizar a táboa de páxinas do proceso.
A page fault service always has to update the page table of the process.
6. **El Working Set de un proceso calculado en sus últimas 1000 referencias a memoria es siempre mayor o igual al calculado en sus últimas 500 referencias a memoria.**
O Working Set dun proceso calculado nas súas últimas 1000 referencias a memoria é sempre maior ou igual ao calculado nas súas últimas 500 referencias a memoria.
The Working Set of a process calculated in its last 1000 memory references is always greater than or equal to the one calculated in its last 500 memory references.
7. **En un sistema con paginación por demanda pura, para que un proceso pueda ejecutar su primera instrucción se necesita prepaginar la página de código que contiene la primera**

instrucción antes de que el proceso pase al estado en ejecución.
Nun sistema de paxinación por demanda pura, para que un proceso execute a súa primeira instrución, a páxina de código que contén a primeira instrución debe ser prepaxinada antes de que o proceso entre no estado de execución.
In a pure demand paging system, in order for a process toexecute its first instruction, the code page containing the first instruction needs to be prepaged before the process enters the running state.
8. **En un sistema de memoria basado en un registro base y un registro límite no puede haber intercambio de procesos a disco.**
Nun sistema de memoria baseado nun rexistro base e nun rexistro límite non pode haber intercambio de procesos a disco.
In a memory system based on a base register and a limit register there can be no process swapping to disk.
9. **El sistema operativo Linux, gestiona la memoria con segmentación paginada.**
O sistema operativo Linux, xestiona a memoria con segmentación paxinada.
The Linux operating system manages memory with paged segmentation.
10. **Una ventaja de las librerías de enlace dinámico con respecto a las librerías de enlace estático es el ahorro de memoria física.**
Unha vantaxe das bibliotecas de ligazóns dinámicas fronte ás bibliotecas de ligazóns estáticas é o aforro de memoria física.
One advantage of dynamic link libraries over static link libraries is physical memory savings.

2. **Los resultado y cálculos o explicaciones tienen que ser correctos para puntuar cada pregunta.** *Os resultados e os cálculos ou explicacións teñen que ser correctos para puntuar cada pregunta.* The results and calculations or explanations have to be correct to score each question.
a) (0.4 puntos) Un sistema de paginación utiliza direcciones lógicas de 16 bits y páginas de 4 Kilobytes. A continuación se muestran las tablas de páginas de dos procesos en ejecución, Proceso 1 y Proceso 2. Solo las entradas y información necesaria para la traducción de direcciones se muestra en la tabla, los bits de permiso y otros no se muestran, y se muestran los valores en decimal. Traduzca las direcciones lógicas a sus direcciones físicas correspondientes.
Un sistema de paxinación usa enderezos lóxicos de 16 bits e páxinas de 4 kilobytes. Móstrase a taboa de páxinas de dous procesos en execución, Proceso 1 e Proceso 2. Na táboa só se mostra as entradas e información necesaria para a tradución de enderezos, os bits de permiso e outros non se mostran. Traduce os enderezos lóxicos aos seus enderezos físicos correspondentes.
A paging system uses 16-bit logical address and 4Kilobytes pages. The following shows the page tables of two running processes, Process 1 and Process 2. Only the entries and information needed for the address translation is shown in the table, permission bits and others are not shown and decimal values are shown. Translate the logical addresses to their corresponding physical addresses.
TP P1

0	3
1	7
2	1
3	5

TP P2

0	2
1	0
2	6
3	4

	Dir lógica en decimal	Dir Física en decimal
P1	11.034	**6.938**
P2	12.345	**16.441**

P1: 11034/4096 = 2, offset 2842.
pag 2 → frame 1, por tanto dir fisica 1x4096 + 2842= 6938

P2: 12345/4096 = 3, offset 57
pag 3 → frame 4, por tanto dir física 4x4096 + 57 = 16441

b) (0.2 puntos) **Si el sistema no tiene memoria virtual, ¿podría ejecutar simultáneamente otros dos procesos P3 y P4, copias de P1 y P2 respectivamente? Si/No, ¿por qué?**
Se o sistema non ten memoria virtual, podería executar simultáneamente outros dous procesos P3 e P4, copias de P1 e P2 respectivamente? Si/Non ¿por qué?
If the system has no virtual memory, could it simultaneously run two other processes P3 and P4, copies of P1 and P2 respectively? Yes/No, why?

Si, si tiene suficiente memoria física instalada y las entradas en la TP tienen bits suficiente para direccionar ese número de frames.

c) (0.2 puntos) **Si el sistema no tiene memoria virtual, ¿podría ejecutar simultáneamente con**

P1 y P2, otro proceso P3 que necesite 20 páginas lógicas para su ejecución? Si/No, ¿por qué?
Se o sistema non ten memoria virtual, podería executarse simultaneamente con P1 e P2 outro proceso P3 que necesite 20 páxinas lóxicas para executarse? Si/Non ¿por qué?
If the system has no virtual memory, could it simultaneously run with P1 and P2 another process P3 that needs 20 logical pages to run? Yes/No, why?

No. Las direcciones lógicas sólo tienen 4 bits para el número de página. Máximo 16 páginas lógicas de un proceso.

d) (0.2 puntos) **Construye una tabla de páginas invertida para un sistema que pueda ejecutar simultáneamente los procesos P1 y P2 de 2a).**
Constrúe unha táboa de páxinas invertida para un sistema que poida executar simultaneamente os procesos P1 e P2 de 2a).
Construct an inverted page table for a system that can simultaneously run processes P1 and P2 from 2a).

	Process ID	Nḿero página
0	2	1
1	1	2
2	2	0
3	1	0
4	2	3
5	1	3
6	2	2
7	1	1

APELLIDOS:_____ **NOMBRE:**_____ **DNI:** _____

Sistemas Operativos (PROCESOS) – Grado Ingeniería Informática UDC. Julio 2023

-Apellidos en MAYUSCULA.

-Examen sin nombre equivale a NO PRESENTADO.

Q1.-(1 punto) Responda Verdadero/Falso (o no conteste) a cada pregunta (0.1 cada una). Respuesta incorrecta -0.1. La puntuación mínima es cero para esta pregunta. (rellenar en el espacio correspondiente de la siguiente tabla).*Answer V/F (True/False) or leave the answer blank. Correct answer: 0.1;Wrong answer: -0.1 (0.1 x 8 = 0.8 points) Minimum score for this question: 0.*Cada unha das preguntas seguintes son de responder V/F (Verdadeiro/Falso) ou non responder. Resposta correcta: +0.1; Resposta incorrecta: -0.1. Puntuación mínima para esta pregunta: 0

1	2	3	4	5	6	7	8	9	10
F	***F***	***F***	***V***	***V***	***V***	***V***	***V***	***F***	***F***

1.-Un sistema Operativo Multiproceso solo puede correr en ordenadores cuyo microprocesador tenga la instrucción "crear proceso".
2.-En un Sistema Operativo Multiusuario, el número de usuarios que puede soportar está limitado por el microprocesador.
3.-El administrador (root) de un sistema puede decidir que un proceso cualquiera se ejecute todo el rato en modo kernel.
4.-Un proceso zombie ocupa una entrada en la tabla de procesos.
5.-La credencial efectiva de un proceso puede cambiar al hacer una llamada exec.
6.-Despues de hacer fork() las credenciales reales y efectivas de los procesos padre e hijo son exactamente las mismas.
7.-Un proceso con la mínima prioridad puede usar el 100% de la CPU si no hay mas procesos listos (runnable) en el sistema.
8.-Desde el punto de vista de tiempo de CPU dedicado a los procesos de usuario un algoritmo no apropiativo es mas eficiente que uno apropiativo.
9.-La instrucción "cambiar prioridad" existe en los microprocesadores actuales y además es una instrucción privilegiada.
10.-En un sistema, todos los procesos creados mediante fork() tienen el mismo conjunto de variables de entorno.

1.-*A Multiprocess Operating System can only run on computers whose microprocessor has the instruction "create process".*
2.-*In a multi-user Operating System, the number of users it can support is limited by the microprocessor.*
3.-*The administrator (root) of a system can decide that any given process run in kernel mode all the time.*
4.-*A zombie process occupies an entry in the process table.*
5.-*The effective credential of a process can change after making an exec call.*
6.-*After doing fork(), the real and effective credentials of the parent and child processes are exactly the same.*
7.-A process with the lowest priority can use 100% of the CPU if there are no other runnable processes in the system.
8.-*From the point of view of CPU time dedicated to user processes, a non-preemptive algorithm is more efficient than a preemptive one.*
9.-*The "change priority" instruction exists in current microprocessors and is also a privileged instruction.*
10.-*On a system, all processes created by fork() have the same set of environment variables.*

1.-Un sistema operativo multiproceso só se pode executar en ordenadores cuxo microprocesador teña a instrución "crear proceso".
2.-Nun Sistema Operativo multiusuario, o número de usuarios que pode soportar está limitado polo microprocesador.
3.-O administrador (root) dun sistema pode decidir que calquera proceso se execute en modo kernel todo o tempo.
4.-Un proceso zombie ocupa unha entrada na táboa de procesos.
5.-A credencial efectiva dun proceso pode cambiar ao facer unha chamada exec.
6.-Despois de facer fork(), as credenciais reais e efectivas dos procesos pai e fillo son exactamente as mesmas.
7.-Un proceso coa prioridade máis baixa pode utilizar o 100% da CPU se non hai outros procesos listos (runnable) no sistema.
8.-Desde o punto de vista do tempo de CPU dedicado aos procesos de usuario, un algoritmo non apropiativo é máis eficiente que un apropiativo.
9.-A instrución "cambiar prioridade" existe nos microprocesadores actuais e tamén é unha instrución privilexiada.
10.-Nun sistema, todos os procesos creados por fork() teñen o mesmo conxunto de variables de ambiente.

Q2.-(0.5 puntos) Queremos que un shell lleve una lista de procesos que ejecuta en segundo plano; en el cuadro vemos la declaración de la lista (implementada como array) y la función de ActualizarListaProcesos, que será invocada para actualizar la lista de procesos en segundo plano. Se supone que el shell mete los procesos (con los valores adecuados en su struct PROCESO) al ser creados como ACTIVOS. En el apéndice puede verse la documentación de las llamadas al sistema y macros utilizadas.

We want a shell to carry a list of processes that it runs in the background; in the box we see the declaration of the list (implemented as array) and the ActualizarListaProcesos function, called by the shell to update the list of background processes. New processes are inserted (with their struct PROCESO correctly filled) as ACTIVE. In the appendix you can see the documentation of the system calls and macros used.

APELLIDOS:_____ NOMBRE:_____ DNI: _____

Queremos que un shell leve unha lista de procesos que executa en segundo plano; no cadro vemos a declaración da lista (implementada como matriz) e a función de actualización da lista. Suponse que o shell inserta os procesos creados (coa sua struct PROCESO recheada correctamente) como ACTIVOS. No apéndice podes ver a documentación das chamadas ao sistema e macros utilizadas.

```
#define ACTIVO 1
#define PARADO 2
#define SENALADO 4
#define TERMINADO 8
#define ALL 15

struct PROCESO {
        pid_t pid;
        uid_t uid;
        int estado;
        int valor;    /*para almacenar el valor o la senal*/
        time_t hora;
        char *linea;
        };

struct LISTAPROC {
        struct PROCESO d[MAXLISTAPROC];
        int fin;
        };
typedef struct LISTAPROC TLISTAPROC;
......
void actualizarProceso (struct PROCESO* p)
{
   int est;

   if (waitpid(p->pid,&est,WNOHANG |WUNTRACED |WCONTINUED)!=-1){
     if (WIFEXITED(est))
        {p->estado=TERMINADO; p->valor=WEXITSTATUS(est);}
     else if (WIFSIGNALED(est))
        {p->estado=SENALADO; p->valor=WTERMSIG(est);}
     else if (WIFSTOPPED(est))
        {p->estado=PARADO; p->valor=WSTOPSIG(est);}
     else if (WIFCONTINUED(est))
        {p->estado=ACTIVO; p->valor=0;}
   }
}
void ActualizarListaProcesos(TLISTAPROC *l)
{
  int i;
    for (i=0; i<l->fin; i++)
                actualizarProceso (&l->d[i]);

}
```

a) Código incorrecto: explicar por qué. Wrong code: explain why. Non é correcto: por qué?

Es incorrecta, waitpid puede devolver
****-1 : error, p.e. el proceso cuyo pid se le pasa no existe***
****el pid que recibe: hay cambios en el proceso que se reporten en el entero est***
****0: se ha llamado con WNOHANG y no tiene nada que informar; no devuelve información en en entero est***
Deberia ser

```
 if (waitpid(p->pid,&est,WNOHANG |WUNTRACED |WCONTINUED)==p->pid){.....
```

De no hacerlo así el programa no actualiza bien los estados de los procesos (se supone que se ha hecho correctamente en prácticas). Ademas, la documentación que se aporta lo pone claramente (zona del apéndice marcada ahora en negrita)

APELLIDOS:_____ **NOMBRE:**_____ **DNI:** _____

Q2.-(0.5 puntos) Sea el siguiente código en C, con todos los includes necesarios y que compila correctamente y que produce un ejecutable a.out. *Consider the following C code, with all the necessary includes and that compiles correctly and the corresponding a.out executable file.* Sexa o seguinte código en C, con todas os includes necesarios e que compila correctamente e que produce un ficheiro executable a.out.

Tanto a.out como f1.txt son del usuario u1, a.out es ejecutado por un usuario u2, desde el mismo directorio donde están a.out y f1.txt. Completar el siguiente cuadro indicando las credenciales reales y efectivas del proceso que ejecuta a.out y si alguno de los descriptores df1 o df2 es -1 dependiendo de los permisos de a.out y f1.txt. *Both a.out and f1.txt are from user u1, a.out is executed by user u2, from the same directory where a.out and f1.txt are. Complete the following table indicating the real and effective user credentials of the process running a.out and if any of the df1 or df2 descriptors is -1 depending on the permissions of a.out and f1.txt.* Tanto a.out como f1.txt son do usuario u1, a.out é executado polo usuario u2, dende o mesmo directorio onde están a.out e f1.txt. Completa a seguinte táboa indicando as credenciais de usuario reais e efectivas do proceso que executa a.out e se algún dos descritores df1 ou df2 é -1 dependendo dos permisos de a.out e f1.txt

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

    int df1, df2;
    df1=open ("./f1.txt", O_RDWR);
    df2=open ("./f1.txt", O_RDONLY);
    printf ("%d, %d\n", df1, df2);
}
```

a.out	f1.txt	ruid	euid	df1=-1?	df2=-1?
rwxrwxrwx	rwxrwxrwx	u2	u2	no	no
rwxr-xr-x	rwxr-xr-x	u2	u2	Si	no
rwxr-xr-x	rwxr--r--	u2	u2	Si	no
rwxr-xr-x	rwsr-xr-x	u2	u2	Si	no
rwxr-xr-x	rwsr--r--	u2	u2	Si	no
rwxr-xr-x	rws------	u2	u2	Si	Si
rwsr-xr-x	rwxr-xr-x	u2	u1	No	no
rwsr-xr-x	rwxr--r--	u2	u1	No	no
rwsr-xr-x	rwsr-xr-x	u2	u1	No	no
rwsr-xr-x	rwsr--r--	u2	u1	No	no
rwsr-xr-x	rws------	u2	u1	No	no

Cuando el ejecutable tiene el permiso rwsr-xr-x se cambia la credencial efectiva a la del propietario del ejecutable, es decir u1.

Para ver si la apertura produce error, se aplican los permisos que corresponden a la credencial efectiva, por tanto cuando la credencial efectiva es u1 se comprueban los permisos de propietario, y cuando es u2, serian los de grupo o los de resto, dependiendo de si u2 es del grupo de f1.txt, pero para f1.txt los permisos de grupo son los mismos que los de resto, que son los que comprobamos para saber si la apertura falla. La apertura en modo O_RDWR necesita permiso de escritura, pero la apertura en modo O_RDONLY sólo necesita permiso de lectura.

APELLIDOS:_____ **NOMBRE:**_____ **DNI:** _____

WAITPID(2) Programmer's Manual WAITPID(2)

NAME

waitpid - wait for process to change state

SYNOPSIS

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *wstatus, int options);
```

DESCRIPTION

All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state (see NOTES below).

If a child has already changed state, then this calls return immediately. Otherwise, they block until either a child changes state or a signal handler interrupts the call (assuming that system calls are not automatically restarted using the SA_RESTART flag of sigaction(2)). In the remainder of this page, a child whose state has changed and which has not yet been waited upon by one of these system calls is termed waitable.

The waitpid() system call suspends execution of the calling thread until a child specified by pid argument has changed state. By default, waitpid() waits only for terminated children, but this behavior is modifiable via the options argument, as described below.

The value of options is an OR of zero or more of the following constants:

WNOHANG
return immediately if no child has exited.

WUNTRACED
also return if a child has stopped (but not traced via ptrace(2)). Status for traced children which have stopped is provided even if this option is not specified.

WCONTINUED (since Linux 2.6.10)
also return if a stopped child has been resumed by delivery of SIGCONT.

If wstatus is not NULL, wait() and waitpid() store status information in the int to which it points. This integer can be inspected with the following macros (which take the integer itself as an argument, not a pointer to it, as is done in waitpid()!):

WIFEXITED(wstatus)
returns true if the child terminated normally, that is, by calling exit(3) or _exit(2), or by returning from main().

WEXITSTATUS(wstatus)
returns the exit status of the child. This consists of the least significant 8 bits of the status argument that the child specified in a call to exit(3) or _exit(2) or as the argument for a return statement in main(). This macro should be employed only if WIFEXITED returned true.

WIFSIGNALED(wstatus)
returns true if the child process was terminated by a signal.

WTERMSIG(wstatus)
returns the number of the signal that caused the child process to terminate. This macro should be employed only if WIFSIGNALED returned true.

WIFSTOPPED(wstatus)
returns true if the child process was stopped by delivery of a signal; this is possible only if the call was done using WUNTRACED or when the child is being traced (see ptrace(2)).

WSTOPSIG(wstatus)
returns the number of the signal which caused the child to stop. This macro should be employed only if WIFSTOPPED returned true.

WIFCONTINUED(wstatus)
(since Linux 2.6.10) returns true if the child process was resumed by delivery of SIGCONT.

RETURN VALUE

waitpid(): on success, returns the process ID of the child whose state has changed; if WNOHANG was specified and one or more child(ren) specified by pid exist, but have not yet changed state, then 0 is returned and the integer pointed by wstatus is not changed.
On error, -1 is returned.

Each of these calls sets errno to an appropriate value in the case of an error.

APELLIDOS:_____**NOMBRE:**_____ **DNI:** _____

USE LETRAS MAYÚSCULAS EN APELLIDOS Y NOMBRE

<<Parte E/S (1.5 puntos)>>. Sistemas Operativos -Grado Ingeniería Informática UDC. Junio de 2023

No se admite entrega de hojas adicionales - - HAY QUE contestar en los espacios proporcionados

1: (0.5 puntos/points)

Tenemos un disco que contiene 65536 (=64*1024) sectores. El disco tiene 4 platos y 2 caras en cada plato. Cada sector contiene 512 bytes, y cada pista contiene 16 sectores. Se ha formateado usando bloques de 4096 bytes.

*Temos un disco que contén 65536 (=64*1024) sectores. O disco ten 4 platos e 2 caras en cada plato. Cada sector contén 512 bytes, e cada pista contén 16 sectores. Formateouse usando bloques de 4096 bytes.*

*We have a disk composed of 65536 (=64*1024) sectors. The disk has 4 platters and there are 2 sides in each platter. Each sector contains 512 bytes, and each track contains 16 sectors. The disk was formatted using blocks of 4096 bytes each.*

A. ¿Cuál es el número total de cilindros del disco? Justifique su respuesta. 512

Cal é o número de cilindros do disco? Xustifique a súa resposta.

Which is the total number of cylinders? Include a brief explanation.

There are 65536 sect * (1track/16sect) = 4096 tracks.
Each cylinder contains 2*4=8 tracks (because there are 4 platters and 2 sides in each of them).
Therefore, there are 4096 tracks * (1cil/8tracks) = 512 cyl.

B. ¿Cuántos bloques se pueden almacenar en el disco? Justifique su respuesta. 8192

Cantos bloques poden ser almacenados no disco? Xustifique a súa resposta.

How many blocks are there? Include a brief explanation.

There are 65536 sect * (1blk/8sect) = 8192 blocks. Note 1 block has 4096 bytes and 1 sect = 512 ➔1 blk occupies 8 sect.

2: (0.25 puntos/points)

El fichero "file.txt" contiene "0123456789\n". El fichero file2.dat no existe.

Complete las líneas 08 y 09 para que el write de la línea 10 escriba “done” en el fichero file2.dat. **NO** es posible volver a llamar a **open()** de nuevo.

*O ficheiro "file.txt" contén "0123456789\n". O ficheiro “file2.dat” non existe. Complete as liñas 08 e 09 para que o write da liña 10 escriba “done” no ficheiro file2.dat. **NON** é posible volver chamar a **open()** de novo.*

*File "file.txt" contains "0123456789\n". File “file2.dat” does not exist. Complete lines 08 and 09 so that the call to write at line 10 outputs “done” into the file “file2.dat”. **NO** more calls to **open()** are allowed.*

```
/* line.01 */  int main(){                              /*program dous23.c*/
/* line.02 */     int i=0,ifd,ofd,bk;
/* line.03 */     bk=dup(STDOUT_FILENO);          // note: STDOUT_FILENO = 1
/* line.04 */     ifd = open("file.txt",O_RDONLY);
/* line.05 */     ofd = open("file2.dat",O_WRONLY|O_CREAT|O_TRUNC, 0666);
/* line.06 */     close(STDOUT_FILENO);
/* line.07 */     dup(ifd);
                  close(STDOUT_FILENO);
/* line.08 */
                  dup(ofd);
/* line.09 */
/* line.10 */     write(STDOUT_FILENO,"done",4);     /**/
/* line.11 */  }
```

3: (0.25 puntos/points)

La cola de peticiones de E/S para acceder a cilindros de un disco contenía las peticiones [165, 190, 33, 40, 250, 200]. Ya se han atendido las peticiones [190] y [200] (en este orden). ¿En qué orden se atenderán las restantes peticiones?

A cola de peticións de E/S para acceder a cilindros dun disco contén as peticións [165, 190, 33, 40, 250, 200]. Xa foron atendidas as peticións [190] e [200] (nesta orde). Indíquese en que orde serán atendidas as restantes peticións.

The I/O requests queue for accessing cylinders within a disk contained the requests [165, 190, 33, 40, 250, 200]. Requests [190] and [200] have already been attended (in this order). Indicate in which order will the pending requests be attended.

Algoritmo / *algorithm*	Cilindros accedidos / *cylinders accessed*			
C-LOOK	250	33	40	165
CSCAN	250	33	40	165

4: (0.5 puntos/points)

El fichero "a.dat" contiene "ABCDEFGH\n". ¿Qué se escribe en pantalla en las llamadas a printf de las líneas lin.11 y lin.14?

O ficheiro "a.dat"contén "ABCDEFGH\n". Que se escribe en pantalla nas chamadas a printf das liñas lin.09 e lin17 ?

File "a.dat" contains "ABCDEFGH\n". What is written into the screen in the calls to printf within lines lin.09 and lin.17 ?

```
//lin.01:   int  main(int argc, char *argv[])   /*program catro23.c*/
//lin.02:   {
//lin.03:    long ret1, ret2;
//lin.04:    char BUF1[8]={'\0','\0','\0','\0','\0','\0','\0','\0'};
//lin.05:    char BUF2[8]={'\0','\0','\0','\0','\0','\0','\0','\0'};
//lin.06:    int fd = open("a.dat", O_RDONLY);
//lin.07:    lseek(fd, 3, SEEK_CUR);
//lin.08:    ret1 = pread(fd, BUF1, 10,7);
//lin.09:    int fd2=dup(fd);
//lin.10:    ret1 = read (fd, BUF1, 2);
//lin.11:    printf("%s--%ld",BUF1,ret1);              /**/
//lin.12:    close(fd);
//lin.13:    ret2 = read (fd2, BUF2, 3);
//lin.14:    printf("%s--%ld",BUF2,ret2);              /**/
//lin.15:   }
```

***Asúmase que no se producen errores durante la ejecución** / *Asúmase que non se producen erros durante a execución* / Assume there were no errors during the execution of the code above.

printf lin.11: =

D	E	-	-	2			

← Contesta aquí / *fill your answer here*
Pon un char en cada celda / *write a char in each cell*

Breve justificación / *breve xustificación / brief explanation* ➔

- lin 07. lseek sets the read offset for file a.dat into position 3 → we will read from 'D' on
- pread() tries to read 10 bytes, from offset 7 on, but only 2 remain (H\n). It is stored into BUF1 → H\n, ret1=2
- lin 10: read() inputs 2 bytes (from offset 3 on) which are stored into BUF1 → DE, ret1=2
- ret1 saves the returned value from read() call; i.e. the number of bytes read→ 2

OUTPUTS: D E - - 2

printf lin.14: =

F	G	H	-	-	3		

← Contesta aquí / *fill your answer here*
Pon un char en cada celda / *write a char in each cell*

Breve justificación / *breve xustificación / brief explanation* ➔

- lin 09. fd2=dup(fd) duplicates the entry fd into fd2. → Both fd and fd2 share the same entry within the Global Open Files Table. Therefore, the changes through fd will also affect fd2.
- lin 10: read() inputs 2 bytes (from offset 3 on) that are stored into BUF1 → DE. The current_pos is moved 2 bytes forward. Therefore, next read() would input data from offset 5 on (i.e. FG...).
- lin 12: closes fd, but fd2 is still active and associated to file “a.dat”.
- lin 13: read() inputs 3 bytes from offset 5 on → FGH, they are stored into BUF2;
- ret2 stores the value returned by read; i.e. the number of bytes read → 3 (ret2=3).

OUTPUTS: F G H - - 3