Apellidos:_____ Nombre:_____ DNI: _____

Sistemas Operativos – Grado Ingeniería Informática UDC. Enero 2024
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 los espacios proporcionados.

Puntuación preguntas: P1: 0.6, P2: 0.3, P3: 0.5, P4:0.6

P1) Un sistema de archivos tipo UNIX tiene un tamaño de bloque de 8Kbytes, i-nodos con 12 direcciones directas, una indirecta simple, una indirecta doble y una indirecta triple. Utiliza direcciones de bloque de 4 bytes. *i*) Calcular e tamaño máximo que podría tener un fichero al usar la indirección doble (sumando lógicamente los bloques de la indirección simple más los 12 bloques de los punteros directos). *ii*) Calcular cuántos bloques de disco son necesarios (en el área de datos) para representar un archivo de tamaño 96 Gbytes + 103 Kbytes. Discriminar cuántos bloques son de datos y cuántos de índices. *iii*) Calcular la fragmentación interna de ese fichero.

Un sistema de ficheiros tipo UNIX ten un tamaño de bloque de 8Kbytes, i-nodos con 12 enderezos directos, un indirecto simple, un indirecto dobre e un indirecto triple. Usa enderezos de bloque de 4 bytes. *i*) Calcular o tamaño máximo que pode ter un ficheiro ao usar a indirección dobre (engadindo loxicamente os bloques da indirección simple máis os 12 bloques dos enderezos directos). *ii*) Calcular cantos bloques de disco son necesarios (na área de datos) para representar un ficheiro de tamaño 96 Gbytes + 103 Kbytes. Discriminar cantos bloques son de datos e cantos de índices. *iii*) Calcular a fragmentación interna dese ficheiro.

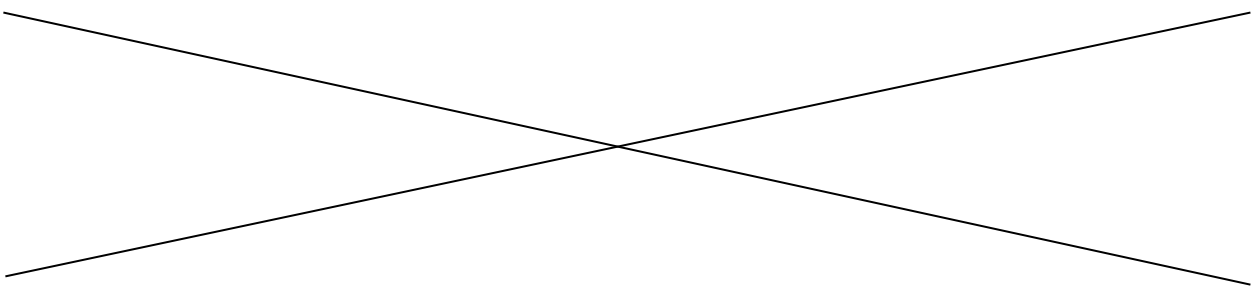
A UNIX file system has a block size of 8Kbytes, i-nodes with 12 direct addresses, a single indirect, a double indirect and a triple indirect address. It uses 4-byte block addresses. *i*) Calculate the maximum size that a file could have when using the double indirection (logically adding the blocks of the simple indirection plus the 12 blocks of the direct pointers). *ii*) Calculate how many blocks are needed (in the data area) to represent a file of size 96 Gbytes + 103 Kbytes. Discriminate how many blocks are data blocks and how many index blocks. *iii*) Calculate the internal fragmentation of that file.

i) Tamaño máximo fichero con i. doble (sumando los bloques con i. simple y los 12 bloques directos)
Tamaño máximo ficheiro con i. dobre (engadindo os bloques con i. simple e os 12 bloques directos)
Maximum size of file with double indirection (adding the blocks with simple indirection and the 12 direct blocks) (0.15p): **32Gbytes + 16 Mbytes + 96 Kbytes**

ii) Nº bloques de datos/Nº bloques de datos/Number of data blocks (0.20p): **12Mbloques + 13**

Nº de bloques de índices/Nº de bloques de índices/Number of index blocks (0.20p): **6149**

iii) Fragmentación interna del fichero/Fragmentación interna do ficheiro/Internal file fragmentation (0.05p): **1024** (bytes)



P2) En ese sistema de archivos UNIX (tamaño de bloque de 8Kbytes), el tamaño de la lista de inodos en disco es de 32 Mbytes. 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. *i*) Calcular el tamaño (en bytes) de un inodo. *ii*) Si la lista de inodos comienza en el bloque lógico 10 (desde el inicio de la partición), ¿qué bloque lógico corresponde al inodo 1285?

Tamaño del inodo: _____ (bytes)
Bloque lógico de inodo 1285: _____

Nese sistema de ficheiros UNIX (tamaño de bloque de 8 Kbytes), o tamaño da lista de inodos no disco é de 32 Mbytes. O superbloque mantén un mapa de bits de inodos libres para determinar os inodos libres/ocupados da lista de inodos. Ese mapa de bits, que forma parte do superbloque, ocupa un total de 8 bloques. *i*) Calcular o tamaño (en bytes) dun inodo. *ii*) Se a lista de inodos comeza no bloque lóxico 10 (desde o inicio da partición), que bloque lóxico corresponde ao inodo 1285?

Tamaño do inodo: _____ (bytes)
Bloque lóxico de inodo 1285: _____

In that UNIX file system (8Kbyte block size), the size of the inode list on disk is 32 Mbytes. The superblock maintains a free inode bitmap to determine the free/busy inodes of the inode list. That bitmap, which is part of the superblock, occupies a total of 8 blocks. *i*) Calculate the size (in bytes) of an inode. *ii*) If the inode list starts at logical block 10 (from the beginning of the partition), which logical block corresponds to inode 1285?

Inode size: **64** (bytes)
Logical block of inode 1285: **20**

P3) Un proceso abre el archivo *"/home/juan/practicas/p1.c"*, cuyo número de inodo es el 5002 y su número de *hard links* inicial es 2. El tamaño del fichero es de 32 Gbytes y se suponen los datos referentes al sistema de ficheros del ejercicio P1 (tamaño bloque, punteros acceso directos e indirectos). Se suponen todos los permisos necesarios en acceso a directorios, apertura del fichero y cambio de sus permisos. Las cachés de datos e inodos están inicialmente vacías. Indicar lo siguiente referente al trozo de código:

Un proceso abre o ficheiro *"/home/juan/practicas/p1.c"*, cuxo número de inodo é o 5002 e o número de *hard links* inicial é 2. O tamaño do ficheiro é de 32 Gbytes e se supoñen os datos referentes ao sistema de ficheiros do exercicio P1 (tamaño bloque, enderezos acceso directos e indirectos). Asúmense todos os permisos necesarios para acceder aos directorios, abrir o ficheiro e cambiar os seus permisos. Os cachés de datos e inodos están inicialmente baleiros. Indique o seguinte sobre o anaco de código:

A process opens the file "*/home/juan/practicas/p1.c*", whose inode number is 5002 and its initial number of *hard links* is 2. The file size is 32 Gbytes and the data referring to the file system from exercise P1 is assumed (block size, direct and indirect access addresses). All necessary permissions are assumed for accessing directories, opening the file and changing its permissions. The data and inode caches are initially empty. Indicate the following regarding the piece of code:

```
int fd1=open("/home/juan/practicas/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/practicas/p1.c", "/home/juan/practicas/practica1.c"); /*crea hard link */ /* hard link is created */
chmod("/home/juan/practicas/practica1.c", 0640);
symlink("/home/juan/practicas/practica1.c", "/home/juan/practicas/p1_slink"); /*crea link simbólico p1_slink */
                                                                              /* symbolic link p1_slink is created*/
int fd2=dup (fd1);

lseek(fd2, 2.000.000, SEEK_SET); /*SEEK_SET indica que el desplazamiento se considera a partir del origen del fichero */
                                 /* SEEK_SET indica que o desprazamento considérase desde a orixe do ficheiro */
                                 /* SEEK_SET specifies that the offset is considered from the origin of the file */
char c=fgetc(fd2);
close(fd1); close (fd2);
unlink("/home/juan/practicas/ practica1.c");
unlink("/home/juan/practicas/p1.c");
```

Cada apartado puntúa 0.1p/ Each question scores 0.1p.

A. ¿Cuál es el valor asignado al descriptor de fichero *fd2*?
   Cal é o valor asignado ao descriptor de ficheiro *fd2*?
   What is the value assigned to the file descriptor *fd2*?: **4**

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

C. Indica los permisos del inodo 5002 en formato *rwxrwxrwx* (después de ejecutar *chmod*):
   Indica os permisos del inodo 5002 en formato *rwxrwxrwx* (despois de executar *chmod*):
   Indicate the permissions of inode 5002 in *rwxrwxrwx* format (after running *chmod*): **rw- r-- ---**

D. ¿Cuál es el número de *hard links* del fichero */home/juan/practicas/p1_slink*?
   Cal é o número de ligazóns duros (*hard links*) do ficheiro */home/juan/practicas/p1_slink*?
   What is the number of *hard links* of file */home/juan/practicas/p1_slink*?: **1**

E. ¿Cuál es el número de *hard links* del inodo 5002 después de ejecutar las dos llamadas *unlink*?
   Cal é o número de ligazóns duros (*hard links*) do inodo 5002 despois de executar as dúas chamadas *unlink*?
   What is the number of *hard links* of inode 5002 after running the two *unlink* calls?: **1**

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

Cada apartado puntúa 0.1p. Cada respuesta errónea puntúa -0.1p. 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 número de aperturas de un fichero se mantiene en el inodo en memoria, no en el inodo en disco (en la la lista de inodos en disco). **T**

B. Al ejecutar la función *sin(x)* de la librería matemática, el proceso está ejecutando exclusivamente en modo usuario. **T**

C. La librería estándar de C (*libC*) incluye el código binario de *unlink()*. **F**

D. Las librerías dinámicas facilitan la actualización de componentes del SO. **T**

E. Un sistema de ficheros con registro (*journaling file system*) disminuye la fragmentación externa. **F**

F. La asociación entre el *uid* (*user indentifier*) del inodo y el nombre simbólico del usuario (propietario del fichero) se encuentra en el fichero */etc/passwd* y puede obtenerse con *getpwuid*. **T**

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 número de aperturas dun ficheiro gárdase no inodo en memoria, non no inodo en disco (na lista de inodos no disco). ____

B. Ao executar a función *sin(x)* da librería matemática, o proceso está executando exclusivamente en modo usuario. ____

C. A librería estándar de C (*libC*) inclúe o código binario de *unlink()*. ____

D. As bibliotecas dinámicas facilitan a actualización dos compoñentes do SO. ____

E. Un sistema de ficheiros con rexistro (*journaling file system*) reduce a fragmentación externa. ____

F. A asociación entre o *uid* (*user indentifier*) do inodo e o nome simbólico do usuario (propietario do ficheiro) atópase no ficheiro */etc/passwd* e pode obterse con *getpwuid.* ____

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 number of opens of a file is kept in the in-memory inode, not in the disk inode (in the disk inode list). ____

B. When executing the function *sin(x)* of the math library, the process is executing exclusively in user mode. ____

C. The standard C library (*libC*) includes the binary code of *unlink()*. ____

D. Dynamic libraries make it easier to update OS components. ____

E. A journaling file system reduces external fragmentation. ____

F. The association between the *uid* (user identifier) of the inode and the symbolic name of the user (owner of the file) is found in file */etc/passwd* and can be obtained with *getpwuid.* ____

Apellidos:_____Nombre:_____ DNI: _____

Examen sin identificación completa NON se califica

Sistemas Operativos - GEI UDC. Xaneiro 2024. Mem+E/S (3 puntos)

Esta parte NON admite entrega follas adicionais. Hay que contestar nas páxinas e espacios proporcionados.

1.

A) (0.5 puntos) Un proceso tiene la cadena de referencias a páginas que se muestra y tiene asignados cuatro marcos de memoria. Las cuatro primeras referencias, estos es, la referencias a las páginas 2, 4, 5 1, producen necesariamente 4 fallos de página porque ninguna página del proceso estaba en memoria. ¿Cual es el número total de fallos de páginas que se producen con el algoritmo de reemplazo FIFO Segunda Oportunidad en las 10 primeras referencias? Obviamente hay que contar esos 4 fallos iniciales en el total. Tanto la asignación de páginas a frames como el total de número de fallos deben ser correctos para puntuar la pregunta.

Un proceso ten a cadea de referencias a páxinas que se mostra e ten asignadas catro marcos de memoria. As catro primeiras referencias, isto é, as referencias as páxinas 2, 4, 5, 1 producen necesariamente 4 fallos de páxina porque ningunha páxina do proceso estaba en memoria. ¿Cal é o número total de fallos de páxina que se producen co algoritmo de reemplazo FIFO Segunda Oportunidad nas 10 primeiras referencias? Obviamente hay que contar esos 4 fallos iniciais no total. Tanto a asignación de páxinas a frames como o total de número de fallos deben ser correctos para puntuar a pregunta.

The string of page references for a process that has been allocated 4 page frames is the one shown below. The first four page references, i.e. references to pages 2, 4, 5, 1, produce 4 page faults because none of the pages of this process were in memory. What is the total number of page faults applying the FIFO Second Chance replacement algorithm after 10 references? Obviously, you have to include the initial 4 faults in the total amount. Both the assignment of pages to frames and the total number of page faults must be right to get the score for this question.

Respuesta: __7__ fallos

Cadena de referencias. String of page references:

2	4	5	1	3	5	6	2	5	3

Asignación de pags a marcos. Asignación de paxs a marcos. Assignment of pages to frames

2	2	2	2	3	3	3	3	3	3
	4	4	4	4	4	6	6	6	6
		5	5	5	5	5	5	5	5
			1	1	1	1	2	2	2
F	F	F	F	F		F	F		

B) (0.5 puntos)

B1. ¿Puede este algoritmo aplicarse con asignación variable de memoria a procesos y reemplazo global? Conteste SI/NO y la razón

B2. ¿Para implementar este algoritmo es necesario el bit de referencia? Conteste SI/NO y la razón

B3. ¿Para implementar este algoritmo es necesario el bit de presencia? Conteste SI/NO y la razón

B4. ¿Para implementar este algoritmo es necesario el dirty bit (bit de modificacion)? Conteste SI/NO y la razón
B5. ¿Para implementar este algoritmo es necesario el lock bit? Conteste SI/NO y la razón
B1. Pódese aplicar este algoritmo con asignación de memoria variable aos procesos e reemplazo global? Responda SI/NON e a razón
B2. É necesario o bit de referencia para implementar este algoritmo? Contesta SI/NON e a razón
B3. É necesario o bit de presenza para implementar este algoritmo? Contesta SI/NON e a razón
B4. É necesario o dirty bit (bit de modificacicón) para implementar este algoritmo? Contesta SI/NON e a razón
B5. É necesario o lock bit para implementar este algoritmo? Contesta SI/NON e a razón
B1. Can this algorithm be applied with variable memory allocation to processes and global replacement? Answer YES/NO and the reason
B2. Is the reference bit necessary to implement this algorithm? Answer YES/NO and the reason
B3. Is the presence bit necessary to implement this algorithm? Answer YES/NO and the reason
B4. Is the dirty bit necessary to implement this algorithm? Answer YES/NO and the reason
B5. Is the lock bit necessary to implement this algorithm? Answer YES/NO and the reason

B1, B2, B3: SI
B4, B5: NO
Explicado todo en las clases de la asignatura.

2. (1 punto)

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

int f1(int f)
{
   int a,b;
   a=4;
   b=a++;
   return(a+b+f);
}

int main(int argc, char *s[], char *t[])
{
  char *d;
  double e;

  e= (double) f1(3);
  d=(char *) malloc(100);
  strcpy(d,"alfa");

   printf ("dir variable almacenada en pila usuario:              %p\n",&d);
   printf ("dir otra variable almacenada en pila usuario:         %p\n",&e);
   printf ("una dir heap:                                         %p\n",&d[0]);
   printf ("otra dir heap:                                        %p\n",&d[1]);
   printf ("num tokens linea comando como entero:                  %d\n",argc);
   printf ("dir segmento código del programa:                     %p\n",f1);
   printf ("otra dir segmento código del programa                 %p\n",f1 +1);
   printf ("nombre del ejecutable como string                     %s\n",s[0]);
   printf ("primera variable de entorno  como string              %s\n",t[0]);
   printf ("dir array punteros a las var entorno                  %p\n",&t);

   free(d);
  exit(0);
}
```

Considere este código. Debe escribir en el espacio reservado 10 sentencias printf para obtener (las direcciones en hexadecimal):

1. dirección de una variable almacenada en la pila usuario
2. dirección de otra variable almacenada en la pila de usuario
3. una dirección del heap
4. otra dirección del heap
5. numero de tokens de la linea comando como entero
6.una dirección del segmento de código del programa
7.otra dirección del segmento de código del programa
8.el nombre del ejecutable como string
9.la primera variable de entorno como string
10.la dirección del array de punteros a las variables de entorno

Considere este código. Debes escribir 10 instrucións printf no espazo reservado para obter (os enderezos en hexadecimal):

1. enderezo dunha variable almacenada na pila de usuarios
2. enderezo doutra variable almacenada na pila de usuarios
3. un enderezo do heap
4. outro enderezo do heap
5. numero de tokens da linea comando como enteiro
6.un enderezo do segmento de código do programa
7.outro enderezo do segmento de código do programa
8.o nome do executable como cadea

9.a primeira variable de ambiente como cadea
10.o enderezo do array de punteiros ás variables de ambiente
Consider this code. You must write 10 printf statements in the reserved space to obtain (the addresses in hexadecimal):
1. address of a variable stored on user stack
2. address of another variable stored on the user stack
3. one heap address
4. another heap address
5. number of tokens of the command line as a integer
6.an address of the program code segment
7.other address of tthe program-code-segment
8.the name of the executable as a string
9.the first environment variable as string
10.the address of the array of pointers to the environment variables

3. (1 punto)

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

int main(int argc, char *s[], char *t[])
 {
   int fda, fdb, pid;
   if ((fda=open("./1.txt", O_CREAT | O_TRUNC | O_RDWR ,0777))==-1){
       perror("No se puede abrir fichero objetivo");
       exit(0);
      }
   fdb=dup(STDOUT_FILENO);
   close(STDOUT_FILENO);
   dup(fda);

   write(STDOUT_FILENO, "1.TXT-CREATED-", 14);

  if ((pid=fork())==-1) {
       perror("Fallo en fork");
       exit(0);
      }
   else if (pid==0)
       write(STDOUT_FILENO, "A", 1);
   else { }

   write(STDOUT_FILENO, "B", 1);

   close(STDOUT_FILENO);
   dup(fdb);

  write(STDOUT_FILENO, "Y", 1);
  write(STDOUT_FILENO, "Z", 1);

  close(fdb);
 exit(0);
}
```

SALIDA POR TERMINAL:

YZYZ

CONTENIDO 1.txt:

1.TXT-CREATED-BAB

En las partes que dos procesos se ejecutan concurrentemente, son posibles otros entrelazados

Este código se compila y ejecuta sin errores, además considere una ejecución en la que las llamadas al sistema no devuelven error. Debe indicar cuál es la salida que se produce por el terminal (o indique NINGUNA si la salida es nula) y el contenido del archivo 1.txt al finalizar la ejecución (o indique VACÍO si existe con 0 bytes, o BORRADO, si ha sido borrado). Finalmente debe dar una explicación de lo que hace el código y por qué produce esa salida, sin esta explicación la pregunta no consigue puntuación.

Este código compila e execútase sen erros, tamén considérase unha execución na que as

chamadas do sistema non devolven erro. Debes indicar que saída produce o terminal (ou indicar NINGUNHA se a saída é nula) e o contido do ficheiro 1.txt ao final da execución (ou indicar VACÍO se existe con 0 bytes, ou ELIMINADO, se é foi borrado). Finalmente, debes dar unha explicación do que fai o código e por que produce esa saída. Sen esta explicación a pregunta non recibe puntuación.
This code compiles and runs without errors, also consider an execution in which the system calls do not return an error. You must indicate what output is produced by the terminal (or indicate NONE if the output is null) and the content of file 1.txt at the end of the execution (or indicate EMPTY if it exists with 0 bytes, or DELETED, if it has been erased). Finally, you must give an explanation of what the code does and why it produces that output. Without this explanation the question does not get score

EXPLICACIÓN

APELLIDOS (mayúscula):_____ NOMBRE:_____DNI. _____

Sistemas Operativos (PROCESOS) – Grado Ingeniería Informática UDC. Enero 2024

-Examen sin identificación completa NO se califica.

-Esta parte NO admite entrega de hojas adicionales. Hay que contestar en los espacios proporcionados.

Q1.-(1 punto) Responda Verdadero/Falso (o no conteste) a cada pregunta (correcta 0.1; 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). 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**	**F**	**F**	**F**	**V**	**V**	**F**	**V**

1-En un Sistema Operativo Multiusuario, el número de usuarios que puede soportar está limitado por el microprocesador.
2.-Sólo un proceso con credencial efectiva del root puede cambiar sus credenciales mediante una de las llamadas exec
3.-Los manejadores de interrupción se almacenan en la pila del kernel
4.-Un proceso zombie no ocupa una entrada en la tabla de procesos.
5.-Las llamadas al sistema exec pueden reemplazar los datos y la pila de un proceso, pero no el código.
6.-Inmediatamente después de su creación, un proceso creado mediante fork() tiene SIEMPRE la misma credencial efectiva que su proceso padre, aunque la real puede no coincidir
7.-Todo proceso es una sucesion de rafagas de CPU y e/s y comienza y termina SIEMPRE (independientemente del S.O.) con una ráfaga de CPU
8.-Un algoritmo apropiativo siempre produce mas cambios de contexto (o como mucho los mismos) que uno no apropiativo.
9.-La prioridad es un entero y su rango de valores depende, en parte, del microprocesador, p.e. una versión de linux para procesadores intel probablemente tenga distintos valores para las prioridades que la misma versión para procesadores superSPARC
10.-En un sistema tipo UNIX, la Tabla de Ficheros Abiertos del sistema es parte de los datos de kernel

1-In a Multiuser Operating System, the number of users it can support is limited by the microprocessor.
2.-Only a process with an effective root credential can change its credentials using one of the exec calls
3.-Interrupt handlers are stored in the kernel stack
4.-A zombie process does not occupy an entry in the process table.
5.-exec system calls can replace the data and stack of a process, but not the code.
6.-Immediately after its creation, a process created using fork() ALWAYS has the same effective credential as its parent process, although the real one may be different
7.-Every process is a succession of CPU and I/O bursts and ALWAYS begins and ends (regardless of the OS) with a CPU burst
8.-A preemptive algorithm always produces more context changes (or at most the same ones) than a non-preemptive one.
9.-The priority is an integer and its range of values depends, in part, on the microprocessor, e.g. a version of Linux for Intel processors will probably have different priority values than the same version for SuperSPARC processors.
10.-In a UNIX-type system, the system's Open Files Table is part of the kernel data.

1-Nun Sistema Operativo Multiusuario, o número de usuarios que pode soportar está limitado polo microprocesador.
2.-Só un proceso cunha credencial root efectiva pode cambiar as súas credenciais usando unha das chamadas exec
3.-Os manexadores de interrupcións almacénanse na pila do núcleo
4.-Un proceso zombie non ocupa unha entrada na táboa de procesos.
5.- As chamadas ao sistema exec poden substituír os datos e a pila dun proceso, pero non o código.
6.-Inmediatamente despois da súa creación, un proceso creado mediante fork() ten SEMPRE a mesma credencial efectiva que o seu proceso pai, aínda que a real pode non coincidir
7.-Cada proceso é unha sucesión de ráfagas de CPU e E/S e SEMPRE comeza e remata (independentemente do SO) cunha ráfaga de CPU
8.-Un algoritmo apropiativo sempre produce máis cambios de contexto (ou como moito os mesmos) que un non apropiativo.
9.-A prioridade é un número enteiro e o seu rango de valores depende, en parte, do microprocesador, por exemplo: unha versión de Linux para procesadores Intel probablemente terá valores de prioridade diferentes que a mesma versión para procesadores SuperSPARC.
10.-Nun sistema tipo UNIX, a táboa de ficheiros abertos do sistema forma parte dos datos do núcleo

APELLIDOS (mayúscula):_____ NOMBRE:_____DNI. _____

Q2.-(0.75 puntos) Se muestra el código de un shell para la ejecución creando proceso en primer y segundo plano. tr es un array terminado a NULL de punteros a caracter con el ejecutable y sus parámetros. Asumimos que un & al final indica ejecución en segundo plano. Se supone además que dicho shell lleva una lista de los procesos que ejecuta en segundo plano (LP). Para cada una de las sentencias marcadas /*UNO*/ /*DOS*) y /*TRES*/ decir si es NECESARIA, INCORRECTA o SUPERFLUA (no es necesaria ni incorrecta, es decir el código funciona tanto con ella como sin ella) justificándolo adecuadamente.

```
ListaProcesos  LP;
........
int ComprobarSegundoPlano(char *tr[])
{
        int i;
        for (i=0; tr[i]!=NULL; i++)
                if (!strcmp(tr[i],"&")){
                        tr[i]=NULL;     /*UNO*/
                        return 1;
                }
        return 0;
}
void Proceso (char *tr[])
{
        int back;
        pid_t pid;
        back=ComprobarSegundoPlano (tr);
        if ((pid=fork())==-1){
              perror ("Imposible crear proceso");
              return;
        }
        if (pid==0){
              if (execvp(tr[0], tr)==-1)
                        perror ("Imposible ejecutar");
              exit(255);   /*DOS*/
        }
        if (!back)
                wait (NULL);    /*TRES*/
        else
                MeterProceso(&LP,pid,tr);
}
```

The table shows the code of a shell for execution, creating processes in the foreground and background . tr is a null-terminated array of character pointers with the executable and its parameters. We assume that a trailing & indicates background execution. It is also assumed that said shell keeps a list of the processes it executes in the background (LP). For each of the statements marked /*UNO*/ /*DOS*) and /*TRES*/ say if it is NECESSARY, INCORRECT or SUPERFLUOUS (it is neither necessary nor incorrect, that is, the code works both with it and without it) justifying it adequately.

No cadro móstrase ocódigo dun shell para a execución, creando procesos en primeiro plano e segundo plano. tr é un array terminada en nulo de punteiros a caracter co executable e os seus parámetros. Supoñemos que un & ó final indica a execución en segundo plano. Tamén se supón que dito shell mantén unha lista dos procesos que executa en segundo plano (LP). Para cada unha das afirmacións marcadas /*UNO*/ /*DOS*) e /*TRES*/ diga se é NECESARIO, INCORRECTO ou SUPERFLUO (non é necesario nin incorrecto, é dicir, o código funciona tanto con el como sen it) xustificándoo adecuadamente.

UNO	**NECESARIA**	**El & es un simbolo del shell para indicar que la ejecución es en segundo plano, si no hacemos tr[i]=NULL, el programa a ejecutar recibe el simbolo & como parámetro, lo cual es incorrecto.**
DOS	**NECESARIA**	**En el caso de que execvp falle (ejecutable no existe, falta de permisos de ejecución, formato incorrecto de ejectutable....), debemos terminar la ejecución del proceso hijo, de lo contrario, cada vez que execvp no pueda ejecutar lo que se le pide, habrá otra copia del shell.**
TRES	**INCORRECTA**	**En el caso de que la ejecución no sea en segundo plano (!back) el shell debe esperar, pero dado que hay procesos en segundo plano, deberá usar waitpid para esperar especificamente por el proceso en primer plano, ya que wait espera por cualquiera, (p.e. uno de los que esté en segundo plano). Lo correcto sería usar *waitpid (pid, NULL, 0)*.**

APELLIDOS (mayúscula):_____ NOMBRE:_____DNI. _____

Q3.-(0.75 puntos) Un sistema tienen una planificación con multiples colas. Hay tres colas: la cola SYS para procesos del sistema, la cola INT para procesos interactivos de los usuarios y la cola BAT para procesos no interactivos. La planificación entre las colas es por prioridades apropiativas, siendo la cola SYS la cola de prioridad más alta y la cola BAT la de prioridad mas baja. La planificación en la cola SYS es un RR de quanto 1, la cola INT lleva un planificación RR de quanto 3 y la cola BAT se planifica por FCFS. Mostrar la planificación de la CPU en el cuadro para los siguientes procesos: A proceso interactivo de usuario, con una rafaga CPU de 7 y llega en el instante 0; B proceso interactivo de usuario, con una ráfaga CPU de 4 y que llega en el instante 1; C proceso del sistema con una ráfaga CPU de 6 y que llega en el instante 6; D proceso del sistema con una rafaga CPU de 3 y que llega en el instante 12.

A system has a schedule with multiple queues. There are three queues: the SYS queue for system processes, the INT queue for interactive user processes, and the BAT queue for non-interactive processes. Scheduling between the queues is by preemptive priorities, with the SYS queue being the highest priority queue and the BAT queue being the lowest priority queue. Scheduling in the SYS queue is a RR of quantum 1, the INT queue carries an RR scheduling of quantum 3 and the BAT queue is scheduled by FCFS. Show the CPU scheduling in the table for the following processes: A user interactive process, with a CPU burst of 7 and arrives at time 0; B interactive user process, with a CPU burst of 4 and arriving at time 1; C system process with a CPU burst of 6 and arriving at time 6; D system process with a CPU burst of 3 and arriving at time 12.

Un sistema ten unha planificación con varias colas. Hai tres colas: a cola SYS para procesos do sistema, a cola INT para procesos interactivos de usuario e a cola BAT para procesos non interactivos. A planificación entre as colas realízase mediante prioridades preventivas, sendo a cola SYS a cola de maior prioridade e a cola BAT a cola de menor prioridade. A planificación na cola SYS é un RR de quanto 1, a cola INT leva unha planificación RR de quanto 3 e a cola BAT está planificada por FCFS. Mostra a planificación da CPU na táboa para os seguintes procesos: Un proceso interactivo de usuario, cunha ráfaga CPU de 7 e chega no instante 0; B proceso de usuario interactivo, cunha ráfaga CPU de 4 e chega no instante1 1; proceso do sistema C cunha ráfaga CPU de 6 e chega no instante 6; Proceso do sistema D cunha ráfaga CPU de 3 e chegando á hora 12.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	**A**	**A**	**B**	**B**	**B**	**C**	**C**	**C**	**C**	**C**	**C**	**D**	**D**	**D**	**A**	**A**	**A**	**B**	**A**

Páginas de manual de wait() and execvp(). wait() and execvp manual pages. Páxinas de manual de wait() e execvp()

```
wait(2)                    System Calls Manual                    wait(2)

NAME
       wait, waitpid, waitid - wait for process to change state

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <sys/wait.h>
       pid_t wait(int *_Nullable wstatus);
       pid_t waitpid(pid_t pid, int *_Nullable 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 ter-
       minated; 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 these calls return immedi-
       ately.  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.
   wait() and waitpid()
       The wait() system call suspends execution of the calling thread until
       one of its children terminates.  The call wait(&wstatus) is equivalent
       to:
           waitpid(-1, &wstatus, 0);
       The waitpid() system call suspends execution of the calling thread un-
       til a child specified by pid argument has changed state.  By default,
       waitpid() waits only for terminated children, but this behavior is mod-
       ifiable via the options argument, as described below.
       The value of pid can be:
       < -1   meaning wait for any child process whose process group ID is
              equal to the absolute value of pid.
       -1     meaning wait for any child process.
       0      meaning wait for any child process whose process group ID is
              equal to that of the calling process at the time of the call to
```

```
exec(3)                  Library Functions Manual                  exec(3)

NAME
       execl, execlp, execle, execv, execvp, execvpe - execute a file

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>
       extern char **environ;
       int execl(const char *pathname, const char *arg, ...
                       /*, (char *) NULL */);
       int execlp(const char *file, const char *arg, ...
                       /*, (char *) NULL */);
       int execle(const char *pathname, const char *arg, ...
                       /*, (char *) NULL, char *const envp[] */);
       int execv(const char *pathname, char *const argv[]);
       int execvp(const char *file, char *const argv[]);

DESCRIPTION
       The  exec() family of functions replaces the current process image with
       a new process image.  The functions described in this manual page are
       layered on top of execve(2). (See the manual page for execve(2) for
       further details about the replacement of the current process image.)
       The initial argument for these functions is the name of a file that is
       to be executed.
       The functions can be grouped based on the letters following the "exec"
       prefix.
   l - execl(), execlp(), execle()
       The const char *arg and subsequent ellipses can be thought of as arg0,
       arg1, ..., argn.  Together they describe a list of one or more pointers
       to null-terminated strings that represent the argument list available
       to the executed program.  The first argument, by convention, should
       point to the filename associated with the file being executed.  The
       list of arguments must be terminated by a null pointer, and, since
       these are variadic functions, this pointer must be cast (char *) NULL.
       By contrast with the 'l' functions, the 'v' functions (below) specify
       the command-line arguments of the executed program as a vector.
   v - execv(), execvp(), execvpe()
       The char *const argv[] argument is an array of pointers to null-termi-
       nated strings that represent the argument list available to the new
       program.  The first argument, by convention, should point to the file-
       name associated with the file being executed.  The array of pointers
```

APELLIDOS (mayúscula):_____ NOMBRE:_____DNI. _____

```
              waitpid().
       > 0    meaning wait for the child whose process  ID  is  equal  to  the
              value of pid.
      The  value  of  options  is an OR of zero or more of the following con-
       stants:
       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.
RETURN VALUE
       wait():  on success, returns the process ID of the terminated child; on
       failure, -1 is returned.

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

       waitid():  returns  0  on  success  or  if WNOHANG was specified and no
       child(ren) specified by id has yet changed state; on failure, -1 is re-
       turned.

       On failure, each of these calls sets errno to indicate the error.
```

```
       must be terminated by a null pointer.
  e - execle(), execvpe()
       The  environment of the new process image is specified via the argument
       envp.  The envp argument is an array  of  pointers  to  null-terminated
       strings and must be terminated by a null pointer.
       All  other  exec()  functions  (which do not include 'e' in the suffix)
       take the environment for the new process image from the external  vari-
       able environ in the calling process.
  p - execlp(), execvp(), execvpe()
       These  functions duplicate the actions of the shell in searching for an
       executable file if the specified filename does not contain a slash  (/)
       character.  The file is sought in the colon-separated list of directory
       pathnames specified in the PATH environment variable.  If this variable
       isn't  defined,  the path list defaults to a list that includes the di-
       rectories returned by confstr(_CS_PATH) (which  typically  returns  the
       value "/bin:/usr/bin") and possibly also the current working directory;
       see NOTES for further details.

       execvpe() searches for the program using the value  of  PATH  from  the
       caller's environment, not from the envp argument.

       If  the specified filename includes a slash character, then PATH is ig-
       nored, and the file at the specified pathname is executed.

       In addition, certain errors are treated specially.

       If permission is denied for a file (the attempted execve(2) failed with
       the  error EACCES), these functions will continue searching the rest of
       the search path.  If no other file is found, however, they will  return
       with errno set to EACCES.

       If  the  header  of  a  file  isn't recognized (the attempted execve(2)
       failed with the error ENOEXEC), these functions will execute the  shell
       (/bin/sh)  with  the  path of the file as its first argument.  (If this
       attempt fails, no further searching is done.)

       All other exec() functions (which do not include  'p'  in  the  suffix)
       take  as  their  first  argument a (relative or absolute) pathname that
       identifies the program to be executed.

RETURN VALUE
       The exec() functions return only if an error has occurred.  The  return
       value is -1, and errno is set to indicate the error.

ERRORS
       All  of  these  functions  may fail and set errno for any of the errors
       specified for execve(2).
```