COGNOMS: **FILA:**

NOM: **COLUMNA:**

Examen final SO. 19 de juny de 2006. Duració: 3 hores

Les notes sortiran el divendres 30 de juny a migdia al racó.
La revisió es farà entre el 3 i el 4 de juliol. El lloc, hora i condicions de les revisions es publicaran al racó amb les notes.
Els problemes 1, 3 i 4 s'ha de lliurar al mateix full de l'enunciat; els problemes 2 i 5 s'han de lliurar en fulls a part, separats.

Pregunta 1 (3 punts)

Contesteu aquesta pregunta a aquest mateix full

a) ¿Per què necessitem una instrucció especial (trap) per entrar al SO i no fem servir un call tradicional?

b) Diferència entre E/S síncrona i asíncrona

c) Quina és la informació mínima que ha de contenir una entrada de directori?

d) Per que l'assignació encadenada de fitxers no és adequada per accessos directes?

e) Al fer un exec, ¿es manté la reprogramació dels signals? ¿Per què?

f) ¿Per què es pot bloquejar un procés al obrir una pipe amb nom per lectura?

g) ¿Com pot ser que es perdin signals?

h) Posa un exemple (amb semàfors) de possible abraçada mortal

i) Què és i per que serveix una taula de pàgines multinivell?

j) Qui actualitza el contingut del TLB?¿Quan ho fa?

COGNOMS: **FILA:**

NOM: **COLUMNA:**

Pregunta 2 (2.5 punts)

Per lliurar en un full a part.

Dado el siguiente conjunto de inodos y bloques de datos de un sistema de ficheros tipo UNIX, del que sabemos que el inodo raíz siempre está en memoria y, que disponemos en memoria de un **buffer cache del tamaño de un bloque**. Se pide:

Tabla inodos

Inodo	2	3	4	8	9	11	12	14
Tipo Fichero	DIR	DIR	DIR	DIR	DATOS	DATOS	DATOS	LINK
Index 1er BD	4	6	7	5	10	9	11	8

Bloques de datos:

Bloque 4		Bloque 5		Bloque 6		Bloque 7		Bloque 8	Bloque 9	Bloque 10	Bloque 11
.	2	.	8	.	3	.	4	/user3/../log	end	top	start
..	2	..	3	..	2	..	3				
Home	3	ln	14	user1	4	log	9				
user3	8	log	12	user2	8						
				log	11						

a) Dibujar el grafo correspondiente a este sistema de ficheros sabiendo que el inodo 2 es el inodo raíz.

b) Indica que accesos a disco son necesarios para abrir el fichero "/home/user2/ln", sabiendo que el fichero no está en uso por ningún otro proceso.

c) ¿Si leemos el fichero del apartado anterior cuál será el contenido?

Se decide realizar la siguiente operación sobre el sistema de ficheros anterior para aumentar la capacidad de almacenamiento secundario, añadiendo una nueva partición de otro disco:

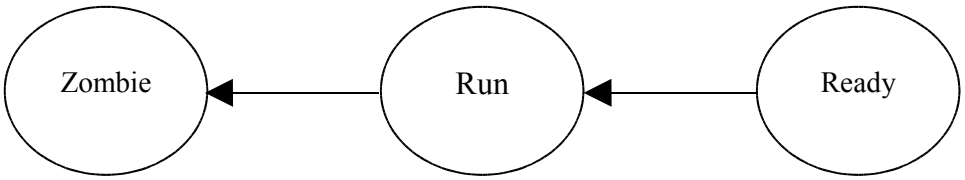
$>mount /dev/hdd2 /home/user2 (monta la segunda partición del disco d en el directorio /home/user2)

d) ¿Si intentamos leer el fichero "/home/user2/ln" cuál será el contenido del fichero leído? Razona tu respuesta.

Pregunta 3 (1 punt)

Contesteu aquesta pregunta a aquest mateix full

Disponemos de un sistema operativo muy básico cuyo grafo de estados de procesos es el siguiente:



Por tanto, un proceso siempre se crea en Ready y cuando pasa al estado de Run, no lo abandona, ni siquiera por entrada/salida, hasta que finaliza.

Responde brevemente a las siguientes preguntas:

a) ¿Cuál es el funcionamiento del siguiente código suponiendo que ninguna llamada al sistema devuelve error?

```
int main ()
{
  switch(fork())
  {
       case 0:  exit(0);
       default: wait(NULL);
  }
  exit(0);
}
```

b) ¿Se necesita algún planificador en este sistema operativo?

c) ¿Cómo se modifica este grafo de estados si añadimos signals a nuestro sistema operativo?

d) ¿Qué característica importante tiene que tener el hardware dónde se ejecuta el sistema para que se pueda enviar un signal desde otro proceso, al proceso que está en Run?

COGNOMS: **FILA:**

NOM: **COLUMNA:**

Pregunta 4 (1 punt)

Contesteu aquesta pregunta a aquest mateix full

Siguin dos processos no emparentats P1 i P2. El procés P1 té 3 threads. El procés P2 té 2 threads.

Siguin A, B, C, D, E, F i G blocs de codi, distribuïts de la següent manera:

P1 Th1	P1 Th2	P1 Th3	P2 Th1	P2 Th2
.	.	.	.	.
.	.	.	.	.
.	B	.	.	.
A	.	.	.	.
.	.	C	.	.
.	.	.	D	E
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	F	.	.	.
.	.	.	.	.
G	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Afegiu el codi de sincronització necessari per tal que:

* A i B s'executin en exclusió mútua

* C S'executi després d'haver acabat A i B

* D s'executi després d'haver acabat C

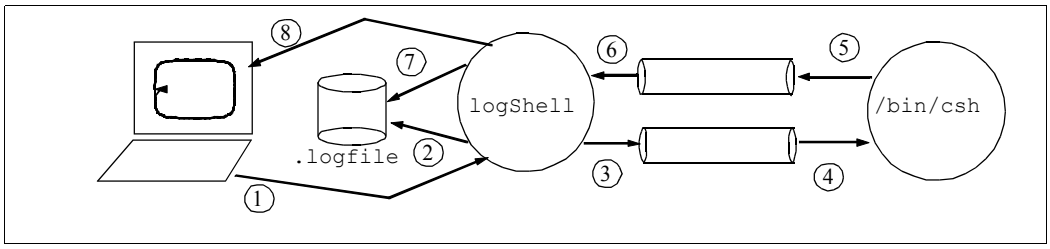
* E s'executi després d'haver acabat D

* G s'executi després d'haver acabat tant E com F

A continuació, indiqueu totes les inicialitzacions necessàries dels mecanismes de sincronització, indicant quin procés fa cada inicialització i en quin moment.

Pregunta 5 (2.5 punts)
Per lliurar en un full a part. Espai màxim: un full per una cara.

En un sistema Unix se quiere mantener en un fichero el registro de todos los comandos que un usuario utiliza durante sus sesiones así como el resultado de los mismos (una copia de lo que el usuario ve en pantalla durante la sesión). Para ello, se escribe un programa (`logShell`), que será el programa inicial del usuario cuando inicie la sesión, que se encargará de crear al proceso que ejecutará el intérprete de comandos (`/bin/csh`). El proceso `logShell` recibe por teclado los comandos que introduce el usuario y los escribe en el fichero de log (llamado `.logfile`) y se los pasa al intérprete de comandos. A continuación, se queda a la espera de recibir el resultado del comando para escribirlo en el fichero llamado `.logfile` y en la salida estándard. La comunicación entre el proceso `logShell` y el intérprete de comandos se hará mediante dos pipes ordinarias: una para que el proceso `logShell` envíe al intérprete de comandos los comandos tecleados por el usuario, y otra para que el intérprete envíe a `logShell` lo que tiene que aparecer en el terminal del usuario. Como simplificación podéis suponer que ni el comando introducido por el usuario ni la respuesta del intérprete ocuparán más de 256 caracteres.



El proceso `logShell` acabará, y por tanto la sesión del usuario, cuando el usuario introduzca el comando `logout`, y por lo tanto el intérprete de comandos acabe la ejecución.
Se pide: implementa el programa `logShell`, en C y usando las llamadas a sistema de Unix para hacer todas las operaciones de Entrada/Salida y de gestión de procesos, suponiendo que no puedes modificar el código del interprete de comandos.
NOTA: No hagas control de errores, ni pongas directivas #include <...>.