APELLIDOS:	**DNI:**
NOMBRE:	
FILA:	**COLUMNA:**

Sistemas Operativos – Facultat d´Informàtica de Barcelona - UPC

Fecha: 9 de Junio de 2008 **Duración: 3 horas**

Notas: Las notas se publicarán el **martes 17 de Junio** en el RACÓ a las 18:00 horas.

Revisión: La fecha, hora y lugar exactos de la revisión del examen se publicarán junto con las notas. Estad atentos.

ATENCIÓN: Las preguntas se tienen que contestar en las mismas hojas de examen utilizando el espacio reservado a tal efecto. Asegúrate de poner NOMBRE y APELLIDOS, DNI, fila y columna en cada una de las hojas. El examen se tiene que entregar en bolígrafo negro o azul. Se puede utilizar el documento con las llamadas al sistema.

Ejercicio 1 (1.5 puntos)

Dibuja el ciclo de vida adecuado para los sistemas operativos con las siguientes características. Además, numera las transiciones e indica en qué casos se producen estas transiciones.

a) Sistema operativo monousuario (un único usuario) y monoprogramado (un único proceso en el sistema).

b) Sistema operativo multiusuario (varios usuarios trabajando a la vez) y monoprogramado (un único proceso por usuario).

c) Sistema operativo multiusuario y multiprogramado (varios procesos por usuario). Los procesos se crean mediante un proceso cargador. En ningún momento saben quien es su proceso padre ni que procesos hijos tienen.

d) Sistema operativo con jerarquía de procesos, gestores de E/S, signals y memoria virtual.

APELLIDOS:	**DNI:**
NOMBRE:	
FILA:	**COLUMNA:**

Exercici 2 (2.5 punts)

Estem en un sistema tipus UNIX amb memòria virtual i paginació, la màquina és monoprocessador. Tenim un sistema de planificació amb prioritats (segon algorisme FIFO) amb apropiació immediata, amb el mateix quantum per tots el processos, i sense envelliment. Suposem que estem executant un procés tot el codi del qual cap en una pàgina (que està a memòria física). Per cadascuna de les següents situacions, contesteu a les preguntes **raonant la vostre resposta.**

Situació a. El procés en RUNNING executa la crida **a=read(h,&c,1)**. La variable h és el resultat d'una crida h=open(...) que no ha retornat -1.

Pregunta: Si el read és d'un fitxer, la crida produirà sempre un accés a disc?

Pregunta: Si el read és d'una *named pipe*, en quins casos el procés es quedarà en RUNNING i en quins passarà a BLOCKED?

Pregunta: Observem que la instrucció read ens torna un -1, però sabem que tot és correcte (és un fitxer de dades que existeix, tenim permissos de lectura, etc...) Què pot haver passat?

Situació b. Arriba una interrupció.

Pregunta: Suposem que és una interrupció de teclat. Al rebre la interrupció, s'executa la rutina d'atenció a la interrupció associada. Això significa que el procés abandona l'estat RUNNING? Justifica la teva resposta.

Pregunta: Suposem que és una interrupció de rellotge. Pot provocar que el procés en RUNNING abandoni la CPU?

Situació c. El programa en RUNNING executa la traducció de la instrucció de llenguatge C **v[i]++;**

Pregunta: indica en quins casos es pot produïr un fallo de pàgina.

Pregunta: hi ha algún cas en que aquesta instrucció pugui provocar que el procés passi a ZOMBIE?

Situació d. Un procés A programa un **alarm(20);** quan han passat 20 segons, el procés A està en blocked. Abans de programar l'alarma, el procés A ha executat la crida **signal(SIGALRM,rutina);**

Pregunta: explica què passa, pas a pas, entre el moment en que el SO nota que han passat 20 segons i el moment en que A acaba d'atendre el signal.

Pregunta: què passaría si en els 20 segons entre l'alarm i el signal, A hagués fet un **fork()**. Qui rebria el signal? El pare? El fill? Tots dos? Justifica la teva resposta.

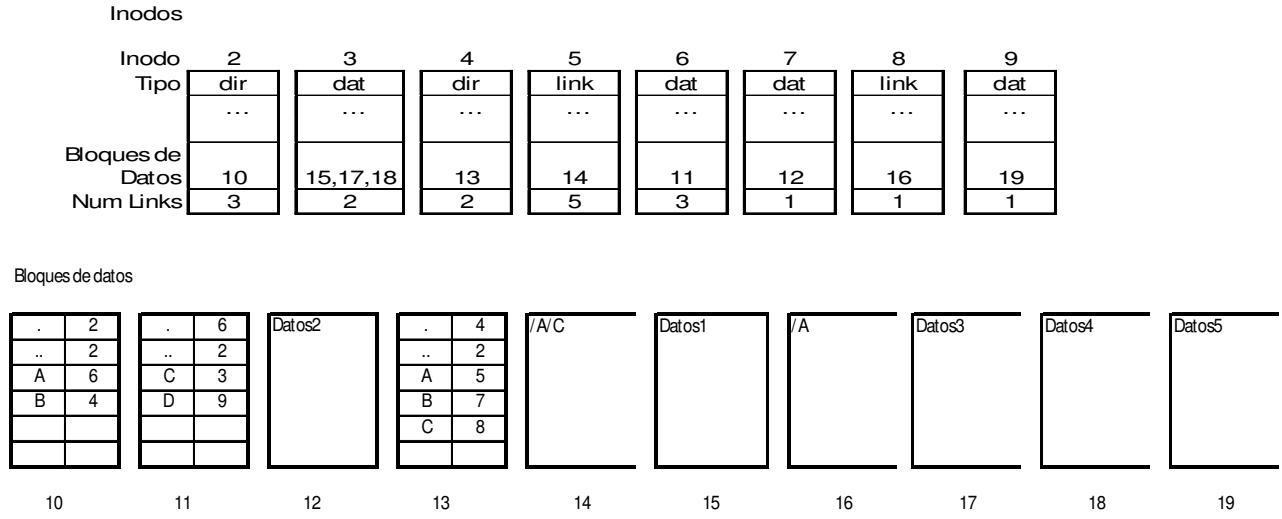
Pregunta: i si en els 20 segons entre l'alarm i el signal el procés A hagués executat una instrucció **execlp("A","A", (char *) 0);**? Justifica la teva resposta.

Situació e. Un dels threads (TH1) del procés en RUNNING es bloqueja en executar un **sem_wait(s);**

Pregunta: Un altre dels threads del mateix procés té una instrucció sem_signal(s); Només un dels threads té aquesta instrucció, i no està dins d'un bucle. Indica tres casos en que el thread TH1 no surt mai de l'estat de bloqueig (mentre el procés és viu, obviament)

Ejercicio 3 (2.5 punts)

Sea el siguiente Sistema de Ficheros de tipo Unix



APELLIDOS:	**DNI:**
NOMBRE:	
FILA:	**COLUMNA:**

a) Indica que errores tiene la estructura anterior (no los señales en el dibujo, indícalos en el espacio que sigue).

b) Dibuja el árbol de directorios que representa la estructura corregida.

c) Supón que los inodos 5 y 8 representan ficheros de datos. Dibuja la FAT y los bloques de datos correspondientes a la estructura anterior corregida.

d) Volviendo al sistema de ficheros Unix. Suponiendo que el tamaño de bloque es de 1KB, que se dispone de una buffer cache de 10 bloques con política LRU, y que un inodo ocupa un bloque, calcula cuántos e indica cuáles accesos a disco son necesarios para cada instrucción de la siguiente sección de código.

```
{
char buffer[250];
…
fd=open (“/B/A”,O_RDONLY);    /*Instrucción 1*/
lseek(fd,2500,SEEK_SET);      /*Instrucción 2*/
read(fd, buffer, 3);          /*Instrucción 3*/
…
}
```

Accesos para Instrucción 1. Cuantos y cuales.

Accesos para Instrucción 2. Cuantos y cuales

Accesos para Instrucción 3. Cuantos y cuales

e) Con las tres operaciones anteriores, ¿Cuántos bytes se transfieren de disco a sistema? Y a la aplicación de usuario, ¿Cuántos bytes se transfieren?

f) Después de las tres operaciones anteriores ejecutamos **close(fd)**, ¿Qué estructuras de proceso y de sistema se ven afectadas por esta operación? ¿Cómo les afecta?

g) El administrador del sistema quiere implantar el anterior SF en un RAID, suponiendo que los discos físicos son de 32GB y que el sistema de ficheros ocupa en total 80 GB: ¿Cuánto discos debe usar para implementar un RAID 0+1? ¿y para un RAID 5? Justifícalo.

h) Además del espacio físico usado, cuál de los sistemas RAID anteriores recomendarías para un SF en el que se harán muchas más escrituras que lecturas. Justifícalo en comparación con el RAID que descartes.

APELLIDOS:	**DNI:**
NOMBRE:	
FILA:	**COLUMNA:**

Ejercicio 4 (2 puntos)

Disponemos de un sistema operativo que implementa named pipes pero no pipes sin nombre. De todas formas, queremos ofrecer al usuario pipes sin nombre que estarán implementadas en una librería. Para implementar estas pipes sin nombre, utilizaremos named pipes.

Se propone la siguiente implementación:

```
int pipe (int pd[2])
{
  pd[0]=open("pipe" O_RDONLY);
  pd[1]=open("pipe", O_WRONLY);
}
```

a) Comenta los problemas de esta implementación

b) Implementa la llamada al sistema pipe, que ahora será una función de usuario, mediante named pipes, solucionando los problemas de la implementación anterior y haciendo que su comportamiento sea, en la medida de lo posible, lo más parecido a la llamada al sistema pipe. **ATENCION**: tened en cuenta que pipe como mucho utiliza dos canales y que las llamadas al sistema tienen que devolver errores!!!

c) ¿Qué desventajas tiene tu implementación respecto a la implementación de pipe como llamada al sistema?

d) Completa el siguiente código en el que un proceso padre se comunica mediante pipes con un hijo al que le envía los números del 1 al 1000.

```
int main()
{
  int pd[2], i;




  switch (fork())
  {
     case 0:




                  while (read(pd[0], &i, sizeof(int))>0);
                  exit(0);
     default:




                  for (i=1; i<=1000; i++)
                    write(pd[1], &i, sizeof(int));




                  wait(&status);
  }
  exit(0);
}
```

e) ¿Esta implementación de la llamada al sistema pipe sirve para poder implementar el código equivalente al comando: ps | grep “a” | wc –l ?

APELLIDOS:	**DNI:**
NOMBRE:	
FILA:	**COLUMNA:**

Exercici 5 (1.5 punts)

Tenim 3 processos (P1, P2 i P3, arribats en aquest ordre). El procés 1 té un únic thread, el procés 2 en té dos threads (TH21 i TH22), i el procés 3 dos threads (TH31 i TH32).

Estem en un sistema de planificació RR amb quantum =4 i prioritats (P1 té prioritat 0 -la més prioritària-, P2 i P3 prioritat 1-la menys prioritària-) i apropiació immediata.

En l'instant inicial, tots els processos estan creats. A l'hora de planificar threads d'un procés, entrarà primer el thread amb el número més petit (és a dir, TH21 abans que TH22), excepte en el cas que l'altre thread hagi estat apropiat: en aquest cas agafarà la CPU el thread apropiat.

El comportament és el següent:

P1 alterna 2 cicles de CPU amb 16 d'e/s.

TH21 alterna 1 cicle de CPU amb 6 d'e/s.

TH22 alterna 4 cicles de CPU amb 5 d'e/s.

TH31 alterna 3 cicles de CPU amb 3 d'e/s.

TH32 alterna 4 cicles de CPU amb 3 d'e/s.

Suposarem que totes les e/s implicades es poden solapar.

Dibuixa el diagrama de Gantt dels primers 20 cicles d'execució.