Ejercicio 1 (2,5 puntos).

Contestar cada una de las preguntas que se hacen a continuación

1. (0,5 puntos) ¿A cuáles de las planificaciones a corto, a medio y a largo plazo, es aplicable la política Round Robin?¿Y la política FIFO / FCFS?

2. (0,5 puntos) Describe las acciones tomadas por el nucleo del sistema operativo para el cambio de contexto entre hilos.

3. (1,5 puntos) Razonar por qué en la práctica del minishell el proceso padre implementa una espera dentro de una estructura condicional while.

```
while (true){

        obtain_order()
        pid = fork();
        switch(pid) {

        case -1:
          perror(" error durante la creación de proceso);
          exit(-1);
         break;
        case 0:
          /* ejecución de mandato */
          execvp(...);
          break;
        default:
         if(bg==0){/*si no hay ejecución en background*/
                while (pid != wait(&estado)){
                        continue; // Provoca una nueva iteración del bucle
                }
    }
}
```

Ejercicio 2. (2,5 puntos).

Tenemos un sistema con direccionamiento de 32 bits, 1GB de memoria física y 8KB de tamaño de página.

a) (0,5 puntos). ¿Cuánta memoria virtual se puede direccionar? ¿Cuántos marcos de página hay?
b) (0,5 puntos). Suponiendo que nuestro sistema operativo genera 1 nivel de tabla de páginas bajo un esquema de asignación de memoria de paginación pura. ¿Cuánto ocupa la tabla de páginas suponiendo que cada entrada ocupa 32 bits?.
c) (1 punto). Suponiendo que hay dos niveles de tabla de páginas y que en la dirección los 9 primeros bits son para acceder a la tabla de páginas del primer nivel. ¿Cuánto ocupan las tablas de páginas de un proceso cuya imagen de memoria ocupa 32Mb?
d) (0,5 puntos). Indica para qué sirven los bits Referenciada, Modificada, Cacheable y Bit de Invalidez de la entrada de la tabla de páginas.

Ejercicio 3. (2,5 puntos).

Tenemos un obsoleto disco duro de 8MB que queremos recuperar del desván para un ordenador donde hemos instalado una versión muy antigua de Unix. Este S.O. formatea el disco del siguiente modo:

Sector de arranque	Superbloque	Mapa de bits	Lista de i-nodos	Bloques de datos

El tamaño de un bloque es 1KB.
El sector de arranque y el superbloque ocupa 1 bloque cada uno.
Tamaño de la dirección de un bloque (2 bytes).
El número de i-nodos disponibles es de 1024. El i-nodo contiene la siguiente información:

- Tipo de archivo y protección. (4 bytes).
- Número de enlaces. (4 bytes).
- Propietario (4 bytes).
- Grupo (4 bytes).
- Tamaño del fichero.(4 bytes).
- Fecha de creación (4 bytes).
- Fecha de actualización (4 bytes).
- Fecha de último acceso (4 bytes).
- 10 entradas de referencias directas a bloque. (2 bytes*entrada)
- 4 entradas de referencia indirecta a bloque. (2 bytes*entrada)
- 2 entradas de referencias indirectas de 2º nive. (2 bytes*entrada).

a) (1 punto). Describe para qué sirve cada campo del formato y del nodo-i.
b) (0,5 puntos). ¿Cuál es el número máximo de directorios que se pueden crear en el sistema de ficheros? ¿Y de archivos?.
c) (0,5 puntos). Calcular el tamaño máximo efectivo del disco después del formateo para los usuarios.
d) (0,5 puntos). Calcular el tamaño máximo teórico de un archivo. (sin contar la limitación de capacidad de este disco duro).

Ejercicio 4 (2,5 puntos).

(2 puntos). Carreras de Hilos (Procesos Ligeros). Escribir un programa completo en C con hilos Posix (pthreads) y variables condicionales que lance mil hilos con el mismo código, estos hilos competirán por ser las primeras en ejecutar su código.

Cada hilo debe:

- Sumar los números desde el 1 hasta el 10000.
- Una vez hecho esto, esperará a que los otros 999 hilos terminen de hacer la suma.
- En ese momento se debe sacar por la salida estándar el mensaje la posición en la que llegaron y terminar.

El hilo principal debe:

- Esperar a que los hilos creados terminen, y entonces terminar el proceso con éxito.
- Lanzar la ejecución de los hilos de una forma consecutiva, numerando a estas según el orden en el que se lanzaron.

(0,5 puntos) Razonar si los hilos terminarán en el mismo orden en el que fueron ejecutados.

Ejercicio 1. Solución:

1. La politica de Round Robin es applicable al planificador a corto plazo. La politica FCFS sería aplicada al planificador a medio plazo y a largo plazo.

2. Un cambio de contexto entre hilos requiere tipicamente salvar el valor de los registros de la CPU del registro que va a dejar de ejecutarse y restaurar los registros de la CPU del nuevo hilo a ejecutar y que se encuentran con su último estado actualizado en el BCP del proceso al que pertenecen.

3. El hecho de tener esta linea : while (pid != wait(&estado)) hace al proceso padre esperar, cuando se ejecuta un mandato en background, no solo por el proceso hijo del mandato actual en ejecución, sino que además provoca la terminación de todos los mandatos previos que se hubieran ejecutado en el minishell en modo background y permanecieran zombies al no haber hecho el minishell ninguna espera (ninguna llamada a wait) por esos procesos. Por lo tanto, en un instante determinado, habrá tantos procesos zombies como mandatos en background se hayan ejecutado consecutivamente.

Ejercicio 2. Solución:

a) Si direcciona 32 bits tenemos 32 bits para la dirección, por tanto una dirección lógica podrá direccionar hasta 2^{32} direcciones = 4GB.

Con 1 GB de memoria física tenemos:

1 GB / 8KB = 2^{30} / 2^{13} = 2^{17} marcos => 128K marcos

b) Para direccionar 8KB de cada página se necesitan 13 bits (2^{13}=8KB). Por tanto quedan 32-13 = 19 bits para direccionar la página. Habrá 2^{19} páginas. Si cada entrada ocupa 32 bits = 4 bytes, tenemos que la tabla de páginas ocupa 2^{19} páginas *2^{2} bytes/entrada = 2^{21} bytes = 2MB

c) Si los primeros 9 primeros bits de la dirección son para acceder a la tabla de primer nivel, y los 13 últimos son para el desplazamiento dentro de la página, nos quedan
32-(13+9) = 10 bits para acceder a la tabla de páginas de 2º nivel. Por tanto, en cada tabla de páginas de 2º nivel habrá 2^{10} entradas y se podrá mantener referenciados hasta 2^{10} marcos = 8MB . Si la imagen de memoria del proceso son 32 MB le hacen falta 4 tablas de segundo nivel (4*8MB = 32MB) + 1 tabla de páginas de 1er nivel.
Cada tabla de páginas de 2do nivel ocupa 2^{10} * 4 bytes = 4KB.
La tabla de páginas de primer nivel tiene hasta 2^{9} entradas por tanto ocupa 2^{9} * 4 bytes = 2 KB.
Por tanto el tamaño total de las tablas de páginas serán 4 tablas de 2do nivel (4KB) + 1 tabla de primer nivel (2KB) = 16KB + 2 KB = 18 KB.

d)

- El bit de Referenciada es activado por la MMU cada vez que una instrucción ejecutada referencia alguna dirección de esa página, y así poder saber si una página fue o no accedida y usarlo para las decisiones de los algoritmos de reemplazo.
- El bit de Modificada es activado cuando se ha producido una modificación sobre los datos almacenados en la página y proceder a su actualización en memoria secundaria cuando la página se seleccione para su reemplazo.
- El bit de Cacheable sirve para indicar en los sistemas que mapean la E/S sobre memoria que esos marcos de memoria pertenecientes a los dispositivos mapeados no deben ser usados en jerarquías de memorias más rápidas.
- El bit de Invalidez indica si la página está presente o no en memoria principal.

Ejercicio 3. Solución:

a)

El sector de arranque contiene la información necesaria para poder iniciar la carga del sistema operativo, es decir, el bloque de carga que lee el iniciador ROM.
El superbloque almacena la metainformación del sistema de archivos. Es decir información que describe cómo se organiza la estructura del sistema de ficheros (tamaño de bloque, número de i-nodos, etc.)
El mapa de bits indica qué bloques de información están ocupados y cuales están libres. Cada bit en la posición i representa con un 0 que el bloque i está libre y con 1 está ocupado.
La lista de i-nodos contiene los descriptores físicos de los archivos (i-nodos) que almacenan las características de cada archivos almacenado.
Por último, los bloques de datos es el espacio reservado para uso efectivo del disco duro.
En cuanto al nodo i, los campos de los que se compone son los siguientes:
Tipo de archivo y protección: El tipo de archivo informa de si el elemento es un directorio, enlace simbólico/físico, fichero regular, etc. La protección indica qué permisos tienen los distintos usuarios sobre las posibles operaciones que se pueden hacer con el archivo.
Número de enlaces. Contador de cuantos enlaces físicos tiene el i-nodo.
Propietario y Grupo tienen el UID y GID que identifica quien es el dueño del archivo y el grupo del dueño.
Tamaño del fichero indica cuánto ocupa en el disco el archivo.
Fecha de creación, Fecha de actualización y Fecha de último acceso son las distintas informaciones relativas al tiempo en que se creó el archivo, en el que se produjo la última actualización y en la que se produjo el último acceso al mismo.
Las entradas de referencias directas a bloque son punteros a los bloques de datos que contiene el fichero. Las entradas de referencia indirecta a bloque son punteros a bloques, donde cada bloque apuntado contiene una o varias referencias a más bloques de datos del fichero. Las entradas de referencias indirectas de 2º nivel añaden otro nivel de indirección para que los bloques de datos de 1er nivel apunten a más bloques de datos del archivo.

b).

El número máximo tanto de directorios como de ficheros será el número máximo de i-nodos disponibles, es decir, 1024.

c)

El tamaño máximo del disco será el tamaño utilizado por los bloques de datos esto es:
8 MB (sector arranque + superbloque + mapa de bits + lista de i-nodos).
El sector de arranque ocupa 1 bloque.
El superbloque ocupa 1 bloque.
El mapa de bits tiene M bits, siendo M el número de bloques del disco. El número de bloques del disco es: 8MB / 1K = 8K Bloques = 8192 bloques. 8192 bits = 1024 bytes = 1 bloque.
Cada i-nodo ocupa 64 bytes. Si hay 1024 i-nodos tenemos 64 bloques para la lista de i-nodos.
Por tanto: 8 Mb (1 bloque + 1 bloque + 1 bloque + 64 bloques) = 8 Mb 67 bloques. En bloques: 8192 67 = 8125 bloques = 8125 K libres para datos de usuario.

d)

El tamaño máximo de un archivo vendrá dado por el número de bloques que se pueden obtener a través de su i-nodo.
Esto es:
10 entradas directas apuntando a bloque => 10 bloques = 10 K
4 referencias indirectas a bloque. Cada bloque puede almacenar hasta 512 entradas a bloque => 512 bloques = 512 K. En total 4 referencias * 512K=2MB.
2 referencias indirectas de 2º nivel. 512 * 512 entradas a bloque por cada referencia de 2º nivel. => 512 * 512 * 2 bloques = 512 K bloques = 512 MB
Por tanto el tamaño máximo será 514 MB con 10K.
Obviamente este tamaño máximo es teórico, puesto que el máximo efectivo no llega a 8MB.

Ejercicio 4. Solución:

```
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#define N 1000
int esperando = 0;
pthread_mutex_t cerrojo = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t espera = PTHREAD_COND_INITIALIZER;

void *trabajador(void *par) {
        int numeroTrabajador = (int) par;
        int i;
        int j;
        int suma = 0;
        for (j=1; j<=10000;j++) suma+=j; /*tarea a realizar por el trabajador*/
        pthread_mutex_lock(&cerrojo);
        if (esperando < N-1) { /* si no han llegado todos me paro a esperarles */
                esperando++;
                pthread_cond_wait(&espera, &cerrojo);
        }
        else{ /* si soy el último en llegar, despierto al resto que me están esperando */
             /* puede usarse también pthread_cond_broadcast(&espera);*/

                for (i=0; i<N-1; i++)
                        pthread_cond_signal(&espera);
        }
                pthread_mutex_unlock(&cerrojo);
                printf ("Soy el numero: %d y mi suma es %d \n",numeroTrabajador,suma);
                pthread_exit(NULL);
}

int main(void) {
        int i;
        pthread_t hilo[N]; /* Identificador de hilo */
        for (i=0; i<N; i++) { /*lanzamiento consecutivo de hilos*/
                if (pthread_create(&hilo[0], NULL, trabajador,(void*)i) != 0) {
                        fprintf (stderr, "Error al crear hilo\n");
                        exit(1);
                }
                sleep(0.01);
        }

        for (i=0; i<N; i++)
                pthread_join(hilo[i], NULL);
        exit(0);
};
```