Nombre: _____ DNI: _____

Apellidos: _____ Grupo: _____

Cuestión 1 Considere un sistema de ficheros basado en i-nodos en el que se utiliza un tamaño de bloque de 1 Kbyte, un tamaño de puntero a bloque de 64 bits, y donde un i-nodo está formado por tres índices directos, 1 índice indirecto simple y 1 índice indirecto doble.

En un directorio de dicho sistema de ficheros existe un fichero con nombre `fichero.txt`. Trabajando en dicho directorio, un usuario escribe y compila los siguientes programas:

```
// programaA
#include ...

int main( void )
{
  int i;
  int fd = open( "fichero.txt", O_RDONLY );
  if( fd == -1 )
  {
    return -1;
  }
  i = lseek( fd, 0, SEEK_END );
  printf( "%d\n", i );
  return 0;
}
```

```
// programaB
#include ...

int main( void )
{
  char c = 'a';
  int fd = open( "fichero.txt", O_WRONLY |
      O_CREAT | O_APPEND );
  if( fd == -1 )
  {
    return -1;
  }
  lseek( fd, 1024, SEEK_CUR );
  write( fd, &c, 1 );
  return 0;
}
```

Tras la ejecución de `programaA`, la salida observada por parte del usuario es la siguiente:

```
$ ./programaA
134144
```

Responda **razonadamente** a las siguientes preguntas (no se valorarán respuestas no razonadas):

a) ¿Cuántos bloques ocupa en disco el fichero `fichero.txt` originalmente?

b) Tras ejecutar una única vez el programa `programaB`, y suponiendo que las rutinas de apertura y escritura no fallan, ¿cuántos bloques de disco ocupará el fichero `fichero.txt`?

c) ¿Cuál será el tamaño del fichero `fichero.txt` (en bytes) en ese momento?

Cuestión 2 En un autoservicio de comida rápida hay cuatro cocineros: dos hacen hamburguesas y otro dos hacen perritos calientes. Cuando un cliente llegue a la barra puede coger o bien una hamburguesa o bien un perrito. Si no hubiera en la barra ninguna unidad del plato elegido por el cliente, éste deberá avisar a todos los cocineros que elaboran ese plato, que podrían estar bloqueados en ese momento. El primer cocinero que despierte repondrá N unidades del plato que cocina y se bloqueará. El segundo simplemente se bloqueará (sin reponer más unidades del plato). Mientras un cliente espera por su plato (por ejemplo, porque no hay hamburguesas) no se debe impedir que otros clientes puedan servirse del otro plato (perritos).

Un número arbitrario de clientes y los cocineros (hilos del programa concurrente) se comportan del siguiente modo:

```
#define N ...

void Cliente(int tipoPlato){
    while (true) {
        conseguirPlato(tipoPlato);
        comer();
    }
}

void Cocinero_hamburguesa(){
    while (true) {
        servirHamburguesa();
    }
}

void Cocinero_perrito(){
    while (true) {
        servirPerrito();
    }
}
```

Implemente las funciones `conseguirPlato()`, `servirPerritos()` y `servirHamburguesas()` empleando mecanismos de sincronización (mútexes, variables condición y/o semáforos) y otras variables compartidas.

Cuestión 3 Considere el siguiente módulo. Asuma que las variables globales `counter` y `Device_Open` valen inicialmente 0, que las funciones `init_module` y `cleanup_module` han sido correctamente implementadas, y que la estructura fops está bien asociada:

```
// ...
static int device_open(struct inode *inode, struct file *file) {
  if (Device_Open) return -EBUSY;
  Device_Open++;
  counter = counter + 1;
  try_module_get(THIS_MODULE);
  return SUCCESS;
}
static int device_release(struct inode *inode, struct file *file) {
  Device_Open--;
  module_put(THIS_MODULE);
  return 0;
}
static ssize_t device_read(struct file *filp, char *buffer, size_t length, loff_t * offset) {
  printk(KERN_ALERT "No soportada.\n");
  return -EPERM;
}
static ssize_t device_write(struct file *filp, const char *buff, size_t len, loff_t * off) {
  counter = counter + 1;
  printk(KERN_ALERT "No soportada.\n");
  return -EPERM;
}
```

Suponga que se ejecuta el siguiente comando, y que el fichero de dispositivo está asociado al módulo anterior:

```
$ ls -lt /dev/ | grep 235
crw-rw-rw-   1 root root     235,  0 ene 18 11:40 dispositivo1
```

a) ¿Cuáles son los números major y minor del fichero de dispositivo? ¿Es de tipo bloque o carácter?

b) ¿Qué orden se ha ejecutado para crear el fichero de dispositivo anterior?

c) ¿Salta algún error al ejecutar `cat /dev/dispositivo1`? ¿De qué función proviene el error?

d) ¿Cuánto valdrá `counter` si tras instalar el módulo y crear el fichero de dispositivo hacemos `echo 123 >/dev/dispositivo1; cat /dev/dispositivo1`?

e) Se desea que, ante una lectura del fichero de dispositivo, se devuelvan al usuario los primeros `length` bytes de la cadena “Esto es una pregunta de ES”, y se muestre el mensaje “Ejecutando operación de lectura” por la salida log del kernel. Proponga una implementación alternativa de alguna de las funciones anteriores para cumplir estas especificaciones.

Cuestión 4 Ejecutamos el siguiente programa en un sistema tipo Unix con memoria virtual, páginas de 4 KB, direcciones de 4 bytes y regiones de texto compartidas:

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <pthread.h>
4  #include <string.h>
5
6  #define BUFFER_SIZE 4096
7  char buffer[BUFFER_SIZE];
8
9  void *producer(void * arg){
10   int index;
11   // producer operations
12   return NULL;
13 }
14
15 void *consumer(void * arg){
16   int index;
17   // consumer operations
18   return NULL;
19 }
20
21 int main()
22 {
23   pid_t pid;
24   FILE *fileOrigin, *fileDestiny;
25   char *nameOrigin = "shakespeare.txt";
26   char *mem_origin, *mem_destiny;
27
28   fileOrigin = open(nameOrigin, O_RDONLY);
29   mem_origin = mmap(0, BUFFER_SIZE, PROT_READ, MAP_SHARED, fileOrigin, 0);
30   if ( (pid=fork()) == -1) {
31     printf("ERROR.\n");
32     exit(-1);
33   }
34   else if (pid==0) {
35     pthread_t th_producer, th_consumer;
36     pthread_create(&th_producer, NULL, (void *)producer, NULL);
37     pthread_create(&th_consumer, NULL, (void *)consumer, NULL);
38     pthread_join(th_producer, NULL);
39     pthread_join(th_consumer, NULL);
40   }
41   else {
42     char *nameDestiny = "copy_shakespeare.txt";
43
44     fileDestiny = open(nameDestiny, O_WRONLY);
45     mem_destiny = mmap(0, BUFFER_SIZE, PROT_WRITE, MAP_PRIVATE, fileDestiny, 0);
46     memcpy(buffer, mem_origin, BUFFER_SIZE);
47     memcpy(mem_destiny, buffer, BUFFER_SIZE); // POINT A
48     fclose(fileDestiny);
49     fclose(fileOrigin);
50     munmap(mem_origin, BUFFER_SIZE);
51     munmap(mem_destiny, BUFFER_SIZE);
52
53     while (wait(NULL) != -1) ;
54   }
55   return 0; // POINT B
56 }
```

Responda a las siguientes preguntas:

a) Rellena la siguiente tabla:

Variable/ Macro	Región Memoria Proceso Padre	Región Memoria Proceso Hijo	Privado/ Compartido	Ejecutable (Si/No)
BUFFER_SIZE				
buffer				
mem_origin				
mem_destiny				
index				

b) Describe para el proceso padre y para el proceso hijo las regiones de la imagen de memoria en el punto A (línea 47) y en el punto B (línea 55).

c) La versión compilada correspondiente al código anterior ocupa 25200 bytes. Las librerías estáticas ocupan 52 KB, no tienen datos inicializados y la región de datos sin valor inicial ocupa 12 KB. ¿Cuántas páginas ocupa el ejecutable?

d) si el tamaño de un char es de 1 byte. ¿Cuál es el tamaño de la región de datos no inicializada?

Cuestión 5 En un sistema GNU/Linux se ejecuta el siguiente programa:

```
1  #include <pthread.h>
2  #include <fcntl.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <stdio.h>
6  #define MAX_THREADS 32
7
8  int fd=-1;
9
10 void* threadfun(void* arg){
11   if (arg==0)
12     fd=open("data.txt", O_WRONLY|
13         O_CREAT|O_TRUNC,0751);
14
15   write(fd,"Hello\n",6);
16   return NULL;
17 }
18
19 int main(void){
20   int i=0;
21   int n=0;
22   pid_t pid;
23   pid_t start=getpid();
24   pthread_t desc[MAX_THREADS];
25
26   threadfun(0);
27
28   for (i=0;i<3;i++) {
29     n+=2;
30     pid=fork();
31     execl("/bin/echo","/bin/echo",
32         "Hello",NULL);
33   }
34
35   while (wait(NULL)!=-1){};
36
37   if (getpid()!=start)
38     exit(0);
39
40   for (i=0;i<n;i++)
41     pthread_create(&desc[i],NULL,
42                         threadfun,1);
43   for (i=0;i<n;i++)
44     pthread_join(desc[i],NULL);
45
46   close(fd);
47   exit(0);
48 }
```

a. Indique (1) cuántos procesos se crean (sin contar al padre original, que es el que invoca la función `main()`), (2) cuántos hilos se crean, (3) cuántos procesos e hilos podrían ejecutarse de forma simultánea cómo máximo, (4) cuál es el contenido del fichero *data.txt* tras la ejecución del programa y (5) qué mostrará el programa por pantalla. Justifique su respuesta. **Nota:** asúmase que todas las llamadas al sistema que invoque el programa tienen éxito.

b. Responda a las preguntas del apartado anterior, pero considerando la siguiente modificación del código: reemplazo de la sentencia de las líneas 31 y 32 (invocación de `execl()`) por lo siguiente: `if (pid==0) write(1,"Hello\n",6);`

Justifique su respuesta, indicando además si el orden de ejecución de los procesos e hilos puede alterar la salida del programa o el contenido final del fichero *data.txt*.