NIA: .....
Nombre y apellidos: .....

Ejercicio 1 (1,5 puntos)

Responda correctamente a las siguientes preguntas y conteste en la siguiente tabla:

1	2	3	4	5	6	7	8	9	10
c	a	d	a	c	d	c	b	c	a

NIA: .....
Nombre y apellidos: .....

Ejercicio 2 (2,5 puntos)

a) Elementos a añadir (o conservar en el caso de rodaja):

- Constantes globales:
  - **#define SETTHRESHOLDMAXSYSCALLS 12345**
  - **#define BLOQUEADO 20**
- Variable global:
  - **long max_syscalls_on**
  - **tipo_listaProcesos listaProcesosDormidos**
- Campos del BCP:
  - **long rodaja, syscalls_on.**

Llamada al sistema nueva:

Llamada al sistema (en espacio de usuario) setThreshold_maxSyscalls (long newMax)

- **R0 = SETTHRESHOLDMAXSYSCALLS**
- **R1 = newMax**
- **TRAP**
- return R0

Llamada al sistema (en kernel) setThreshold_maxSyscalls ()

- **if (R1 >= 0)**
  - **max_syscalls_on = R1**
  - R0 = 0
- **return // return from interruption**

Elementos existentes en el S.O. a modificar:

Pseudocódigo Llamada a crearProceso() // fork()

- **newBCP->syscalls_on = 20L;**
- **newBCP->rodaja = RODAJA;**
- <código original de crearProceso>

Pseudocódigo arrancar_sistema_operativo ()

- **max_syscalls_on = 0L;**
- **listaProcesosDormidos = crearListaVacia() ;**
- <código original de arranque del sistema>

NIA: .....
Nombre y apellidos: .....

Pseudocódigo Manejador_Llamadas_sistema()

- **procesoActual->syscalls_on ++;**
- <Código anterior del manejador>

Pseudocódigo Manejador_interrucion_reloj()

- Ticks = Ticks +1;
- Insertar_Interrupcion_Software(**TratarRodaja2**)
- Generar_Interrupcion_Software();

Pseudocódigo TratarRodaja2()

- **procesoAnterior = primero(listaProcesosDormidos)**
- **mientras (procesoAnterior !=NULL) && (procesoAnterior->rodaja >= Ticks)**
  - **procesoAnterior->estado = LISTO**
  - **procesoAnterior->rodaja = RODAJA**
  - **InsertarAlFinal(procesoAnterior, listaProcesosListos)**
  - **borrarPrimero(listaProcesosDormidos)**
  - **procesoAnterior = primero(listaProcesosDormidos)**
- **if (procesoActual->syscalls_on >= max_syscalls_on) && (max_syscalls_on != 0L)**
  - **InsertarAlFinal (procesoActual, listaProcesosDormidos)**
  - **procesoActual->estado = BLOQUEADO**
  - **procesoActual->rodaja = Ticks + 5 * RODAJA**
  - **procesoActual->syscalls_on = 0L**
  - procesoAnterior = procesoActual
  - procesoActual = planificador()
  - procesoActual->estado = EJECUTANDO
  - CambiarContexto (procesoAnterior, procesoActual)
- procesoActual->rodaja = procesoActual->rodaja - 1
- Si (procesoActual->rodaja == 0)
  - **InsertarAlFinal (procesoActual, listaProcesosListos)**
  - **procesoActual->estado = LISTO**
  - **procesoActual->rodaja= RODAJA**
  - **procesoActual->syscalls_on = 0L**
  - procesoAnterior = procesoActual
  - procesoActual = planificador()
  - procesoActual->estado = EJECUTANDO
  - CambiarContexto (procesoAnterior, procesoActual)

NIA: .....
Nombre y apellidos: .....

Ejercicio 3 (2 puntos)

a)

Permite que el sistema operativo pueda ejecutar otro proceso mientras espera a que una tecla sea pulsada, y sea copiada a memoria.

b)

Fragmento	Descripción
1	(int. software) Despertar a un único proceso
2	(petición/ll. sistema) realizar un cambio de contexto (voluntario).
3	(int. hw) Generar una int. sw

c)

Funcionalidad	Descripción
Int. Hardware	Insertar_tecla(tecla, Teclado.BufferTeclas) Fragmento 3
Int. Software	Fragmento 1 cambioContexto(procesoAnterior, procesoActual)

read(fd,buffer,size)	mientras (estaVacio(Teclado.BufferTeclas)) {
	Fragmento 2
	}
	Buffer[0] = extraerTecla(Teclado.BufferTeclas)
	return 1

NIA: .....
Nombre y apellidos: .....

Ejercicio 4 (2 puntos)

a) Completar el diseño de estructuras en disco:

Superbloque

- **Int Número mágico, inicialmente a 0x1234**
- **Int Tamaño de bloque, inicializado a 4096 // R1**
- **Int Número de bloques, inicializado a 50 // R2**
- **Int Tamaño del inodo, inicializado a 128 // R6**
- **Int Número de inodos, inicializado a 21 // R4**
- **Char bloques_estado[50], inicialmente todos a '0' (menos los tres primeros) // R8**
- **Char relleno[] // R3 (no directorios) y R7**

Inodo

- **Char estado, inicialmente a '0' // R6**
- **Char nombre[10], inicialmente a "" // R6**
- **Int bloques[5], inicialmente a cero ambos // R5**
- **Int tamaño, inicialmente a 0**
- **Char relleno[] // R3 (no directorios -> no más campos)**

NIA: .....
Nombre y apellidos: .....

b) Diseño de estructuras en memoria:

Superbloque sbloque
Inodo inodos[21]
Sesion sesiones[21]

Donde Sesión es una estructura con, al menos, los siguientes campos:

- **Int posición[0..100] // R9**
- **Int indice_posicion // R9**
- **Int abierto**

c) bmap:

```
int bmap ( int inodo_id, int offset )
{
    int bid = offset / sb.tamanyo_bloque;
    if (bid > 4)
        return -1;

    return inodos[inodo_id].bloques[bid];
}
```

NIA: .....
Nombre y apellidos: .....

d) ialloc, ifree y namei:

```
int ialloc ( void )
{
   for (int=0; i< sb.numero_de_inodos; i++)       // en toda la tabla de inodos…
   {
       if (inodos[i].estado == '0') {              // si inodo está marcado como libre
          memset(&(inodos[i]), 0, sizeof(inodo)); // poner valores por defecto
          inodos[i].estado = '1';                 // ocupar inodo
          return i;                               // devolver identificador
       }
   }
   return -1;                                     // si no hay inodo libre, devolver -1
}


void ifree ( int inodo_id )
{
    inodos[inodo_id].estado = '0';                // liberar i-nodo
}


int namei ( char *fname )
{
    for (int=0; i< sb.numero_de_inodos; i++)      // buscar inodo con nombre <fname>
    {
        if (! strcmp(inodos[i].nombre, fname))
            return i;                             // devolver identificador
    }
    return -1;                                    // si no está el inodo, devolver -1
}
```

NIA: .....
Nombre y apellidos: .....

e) **Posibles problemas:**

De cara a ahorrar espacio en metadatos:

- Usar un único bloque de disco para inodos, o si fuera posible, usar un único bloque de disco para el subperbloque e inodos.
- No usar un mapa de bytes sino de bits.

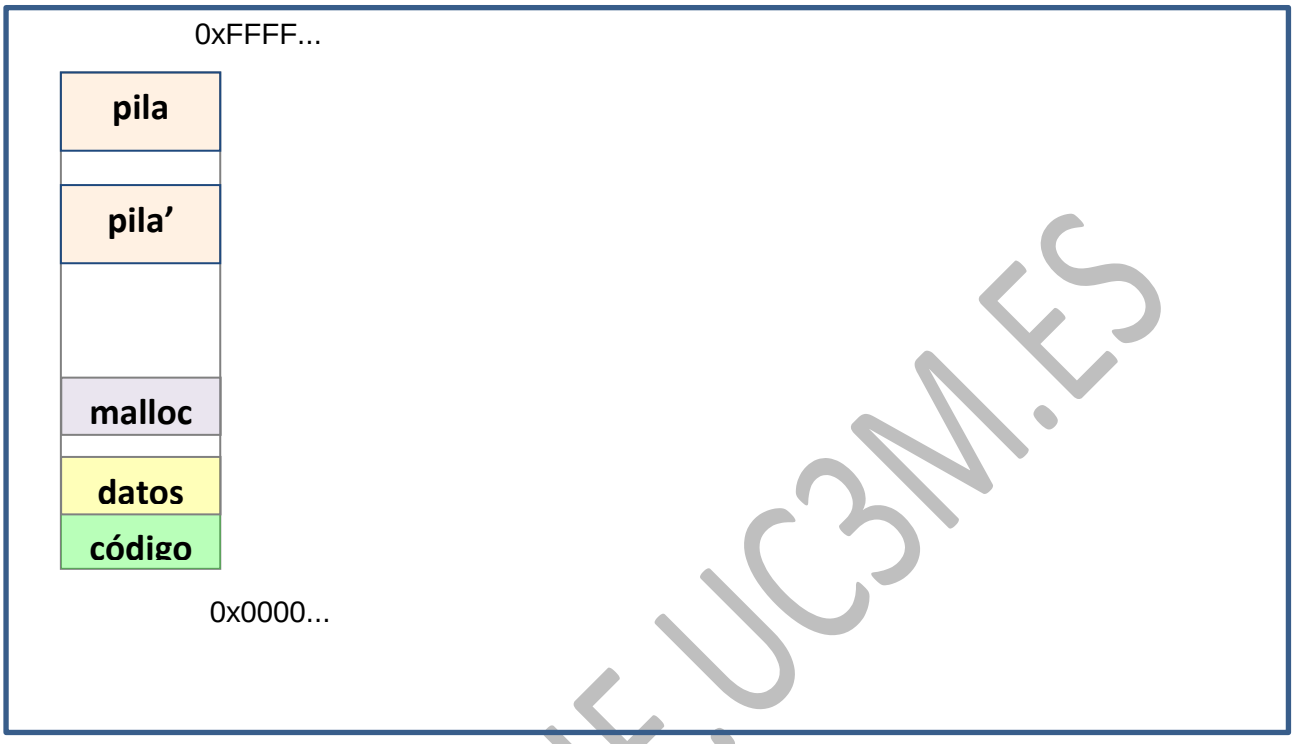
De cara a eficiencia:

- Gestionar los inodos disponibles en una matriz independiente de cada inodo, ya que eso permite no tener que traer a memoria principal todos los inodos para saber cuál está libre.
  Esta operación es necesaria al tener la contabilidad de libre/ocupado en cada inodo y buscar un inodo libre me obliga a tener todos los inodos en memoria.
  Además el direccionamiento sobre dirección base + desplazamiento en un vector es más rápido y requiere de menos instrucciones máquina que el continuo direccionamiento para llegar a cada inodo y luego desplazarse a ese campo

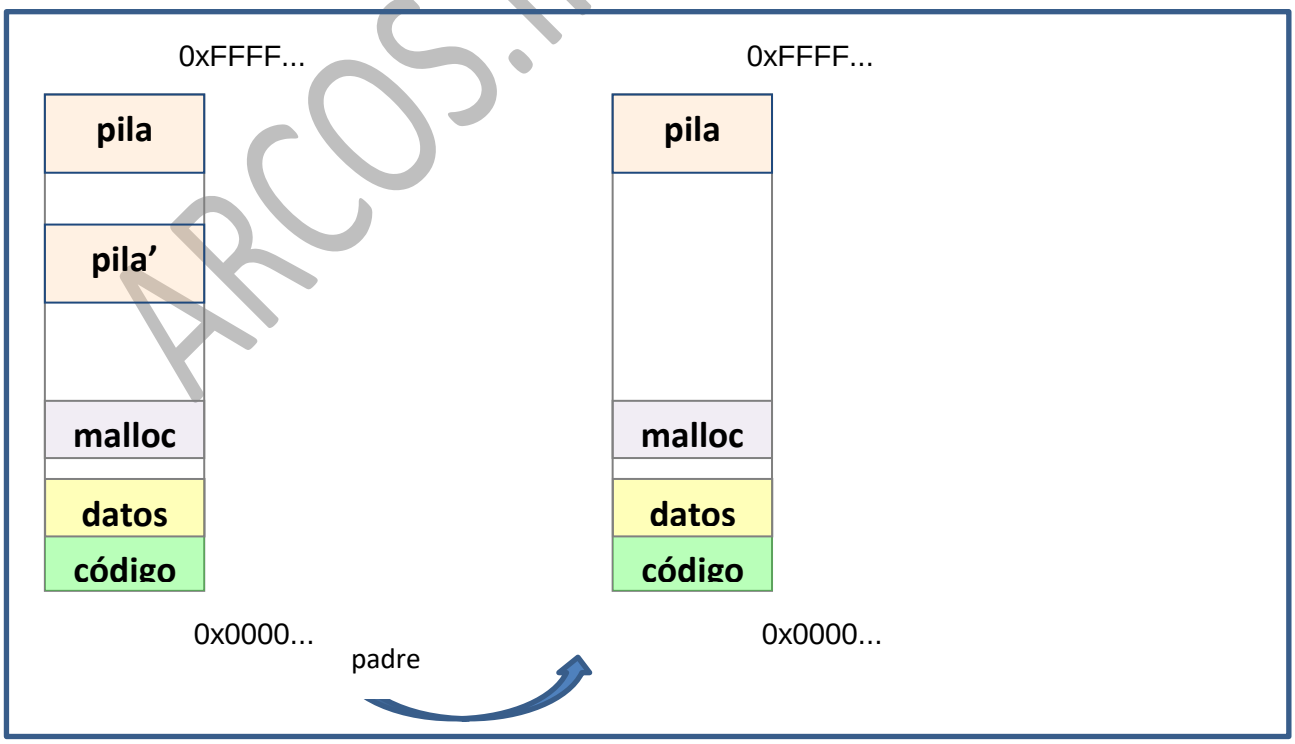
NIA: .....
Nombre y apellidos: .....

Ejercicio 5 (2 puntos)

a)

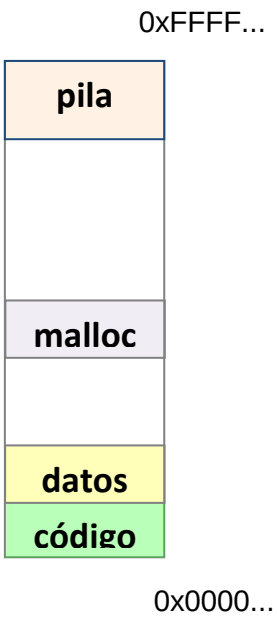


b)



NIA: .....
Nombre y apellidos: .....

c)



d) Si, que arg sobre la que se hace malloc es una variable local, por lo que cuando se deje de ejecutar la función se perderá el acceso a la zona de memoria pedida.
De igual forma pint nunca se llega a usar.
Lo ideal sería pasar pint por referencia, y usar (*(int **)arg) en lugar de arg en task1.