Nombre:
DNI:

Segundo control de laboratorio

Crea un fichero que se llame "respuestas.txt" donde escribirás las respuestas para los apartados de los ejercicios del control. Indica para cada respuesta, el número de ejercicio y el numero de apartado (por ejemplo, 1.a).

Justifica brevemente todas tus respuestas. Una respuesta sin justificar se considerará como no contestada.

Importante: para cada uno de los ejercicios tienes que partir de la versión de Zeos original que te hemos suministrado.

1. (6 puntos + 1 punto de implementación) to task or not to task...

Queremos implementar un mecanismo de creación de tareas en ZeOS. Una tarea es similar a un thread visto en clase (comparte el espacio de direcciones del proceso padre) con la salvedad que permite ejecutar una función con un parámetro y no se necesita especificar ninguna pila. El sistema se encarga de asignar una zona de memoria para usar como pila donde ejecutar esta función.

Tienes que implementar la siguiente llamada a sistema:

*int createTask(void (*f)(int), int parameter)*

que creará una nueva tarea para ejecutar la función *"f(parameter)"*.
El sistema debe crear una pila (para simplificar vamos a suponer que las pilas tienen un tamaño fijo de 4Kb) y asignarla a esta tarea. Como en el caso de los threads, esta pila debe ser accesible en el espacio de direcciones del proceso que crea esta tarea y situarse en algun lugar disponible al final de este espacio.
Como resultado devuelve el identificador de la tarea o -1 si ha ocurrido algún error, actualizando la variable global *errno* con el tipo de error concreto.
Una tarea puede hacer copias de si misma con *fork*, pero no debes permitir que cree threads. Puedes suponer que la gestión de memoria dinámica NO está implementada.
Ejemplo de uso, en el que se crean 4 tareas que muestran el valor pasado como parámetro:

```
void f(int i) {
        char b[10];
        write(1, "Soy ", 4);
        itoa(i, b);
        write(1, b, strlen(b));
        exit();
}
int main() {
        int i;
        for (i=0; i<4; i++)
                createTask(f, i);
        while(1);
}
```

Nombre:
DNI:

a) (0.25 puntos) Si ejecuto *f(42)* en el *main*, muestra el contenido de la pila, justo antes de ejecutar primera instrucción de ensamblador de la función ´f´.
b) (0.25 puntos) Muestra la línea de código ensamblador de la función ´f´ del ejemplo que accede al parámetro ´i´.
c) Indica el código de la función *findEmptyTaskStack* que, a partir de una tabla de páginas, encuentra un espacio disponible al final del espacio de direcciones para guardar la pila de la tarea y devuelva su dirección inicial.
d) Indica el código de la función *mapTaskStack* que, a partir de una tabla de páginas y una dirección de memoria, reserva espacio físico para la pila y lo mapea en la dirección.
e) Indica el código de la función *prepareTaskStack* que a partir de una dirección de memoria y de una variable entera ´i´, configura esta zona de memoria como una pila en la que hay guardada el valor ´i´ y devuelva una dirección que puede ser usada como pila por el código de una función que recibe un parámetro, como la función ´f´ del ejemplo.
f) (0.25 puntos) Indica que campos nuevos son necesarios en el PCB y justifica cada uno.
g) (0.5 puntos) ¿Qué cambios son necesarios en la llamada a sistema *fork*?
h) (0.75 puntos) ¿Hay que cambiar alguna otra rutina existente?
i) Completa el pseudocódigo de *sys_createTask:*

```
int sys_createTask(void (*f)(int), int i) {
        [1]
    "x = sys_clone( [2], [3] );" pero sin llamarla directamente
        [4]
}
```

j) Implementa en Zeos esta nueva funcionalidad. Puedes usar el mecanismo que creas conveniente para añadir la nueva llamada a sistema.

2. *(2 puntos)* Threads

a) ¿Cómo puedes determinar si un PCB cualquiera es un thread de un proceso?
b) En qué pila nos encontramos cuando se ejecuta el *ret_from_fork* del *sys_clone*, ¿en la pila de usuario o en la pila de sistema?

3. *(2 puntos)* Read

a) ¿Tiene alguna importancia bloquear el proceso al principio o al final de la cola de bloqueados?
b) Supón el caso que el buffer circular esta vacio y un proceso hace un *read*. Posteriormente, en el momento en que se produce la 1a interrupción de teclado, ¿qué proceso estará en ejecución?