SOA (18/12/2018)

Nombre:
DNI:

Segundo control de laboratorio
Grupo 12

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 número de apartado (por ejemplo, 1.a).

Importante: para el primer ejercicio tienes que partir de la versión de Zeos original que te hemos suministrado.

1. (4 puntos) Zeos Scheduler

Queremos modificar la llamada al sistema fork para cambiar su comportamiento. La nueva cabecera de esta llamada será:

```
int fork(int time);
```

Donde *time* es el tiempo mínimo, en ticks de reloj que tiene que pasar antes de que se ejecute por primera vez el proceso hijo. Cuando pasa este intervalo de tiempo, se tiene que forzar un cambio de contexto al proceso hijo. Después de este primer cambio de contexto, se procederá con el comportamiento por defecto del planificador para este proceso. Si hay más de un proceso que se tiene que ejecutar en el mismo instante, se elegirá al primero que se creó y el resto se retrasarán un tick de reloj.

En el caso de que *time* sea igual o menor que 0, se ejecutará el comportamiento por defecto que tiene fork (el hijo se encola en la cola de Ready hasta que el planificador decida ejecutarlo).

Por supuesto, se valorará que la solución sea lo más efectiva y eficiente posible.

Responde a los siguientes apartados:

a) Indica que nuevas estructuras se tienen que añadir al sistema operativo para implementar esta llamada.
b) Indica qué modificaciones se tienen que hacer a las estructuras actuales del sistema para implementar esta llamada.
c) Indica qué modificaciones se tienen que hacer en sys_fork.
d) Indica desde dónde se tiene que hacer la comprobación de que ha pasado el intervalo de tiempo especificado en fork.
e) Indica el código que fuerza el cambio de contexto cuando ha pasado el intervalo de tiempo especificado en fork.
f) (1,5 puntos) Implementa en Zeos el mecanismo descrito.

Nombre:
DNI:

2. (6 puntos) Sockets

Queremos implementar un *port scanner*. Éste es un proceso que comprueba si los puertos de una máquina, cuyo rango se pasa a través de la línea de comandos, están abiertos o cerrados. La línea de comandos que se utiliza para ejecutarlo es:

>port_scanner machine_ip 4500 6000

En este caso, comprobaría si la máquina cuya dirección ip es *machine_ip*, tiene abiertos los puertos comprendidos en el rango [4500, 6000] (ambos inclusive).

Esta implementación tiene que ser multihilo y cada uno de los hilos tiene que escanear aproximadamente el mismo número de puertos. El número de hilos que creará será como máximo 10.

El resultado será una lista *ordenada* por número de puerto, indicando si un puerto está abierto o cerrado.

Responde a los siguientes apartados:

a) Indica el conjunto de **llamadas al sistema** que tiene que realizar *port_scanner* para decidir si un puerto está abierto o cerrado.
b) Indica el algoritmo para calcular el número de hilos que se tienen que crear junto con el cálculo del rango de puertos que tiene que escanear cada hilo.
c) Escribe el pseudocódigo del bucle principal de escaneo de puertos de cada uno de los hilos.
d) Explica el mecanismo para conseguir que la lista de puertos escaneados sea ordenada teniendo en cuenta que los hilos no tienen por qué ejecutarse a la misma velocidad.
e) En tu opinión, ¿la implementación que se propone será más eficiente comparada con una implementación multiproceso concurrente del servidor en vez de multihilo?
f) (2 puntos) Implementa el programa descrito.

3. (1 punto) Generic Competences Third Language (Development Level: mid)

The following paragraph belongs to the book *Understanding the Linux Kernel* by D. Bovet and M. Cesati:

"Memory allocation requests can be satisfied in two different ways. If enough free memory is available, the request can be satisfied immediately. Otherwise, some memory reclaiming must take place, and the kernel control path that made the request is blocked until additional memory has been freed. However, some kernel control paths cannot be blocked while requesting memory— this happens, for instance, when handling an interrupt or when executing code inside a critical region. In these cases, a kernel control path should issue atomic memory allocation requests. An atomic request never blocks: if there are not enough free pages, the allocation simply fails."

Nombre:
DNI:

Create a text file named “generic.txt” and answer the following questions (since this competence is about text understanding, you can answer in whatever language you like):

1. How can a memory request be satisfied?
2. In which situations a memory request will be considered as critical?
3. In a critical memory request, what is the default behavior of the kernel if not enough memory is available?

4. Entrega

Sube al Racó los ficheros “respuestas.txt”, “generic.txt” y el código que hayas creado en cada ejercicio.

Para entregar el código, si lo has hecho, de cada ejercicio utiliza:

```
> tar zcfv dni.tar.gz directorio_ejercicio1 directorio_ejercicio2
```