DISEÑO DE SISTEMAS OPERATIVOS

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 2

Driver de dispositivo de bloques

Realizado por:

David del Rey García 100315243

(100315243@alumnos.uc3m.es)

Óscar Hernández Muñoz-Reja 100317624

(100317624@alumnos.uc3m.es)

Rodrigo Borges 100317579

(100317579@alumnos.uc3m.es)

Contenido

1. Introducción

En el presente documento se pretende explicar y detallar las mejoras realizadas al driver del disco duro emulado a través de *disk.dat*. Los driver son elementos software que permiten la comunicación de distintos dispositivos con un ordenador. En nuestro caso el software permite la comunicación entre un disco duro y el ordenador de manera óptima.

Este documento está estructurado de la siguiente manera.

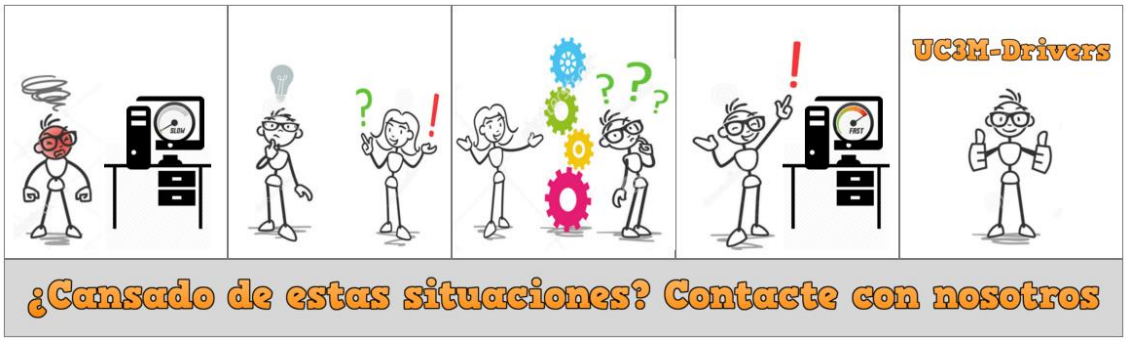
En primer lugar tenemos una descripción de cada una de las mejoras propuestas, con la cuáles se pretende mejorar el actual rendimiento del driver.

A continuación tenemos la parte de las pruebas. En esta sección se podrá ver de forma detallada y gráfica el aumento de rendimiento que han supuesto las mejoras implementadas.

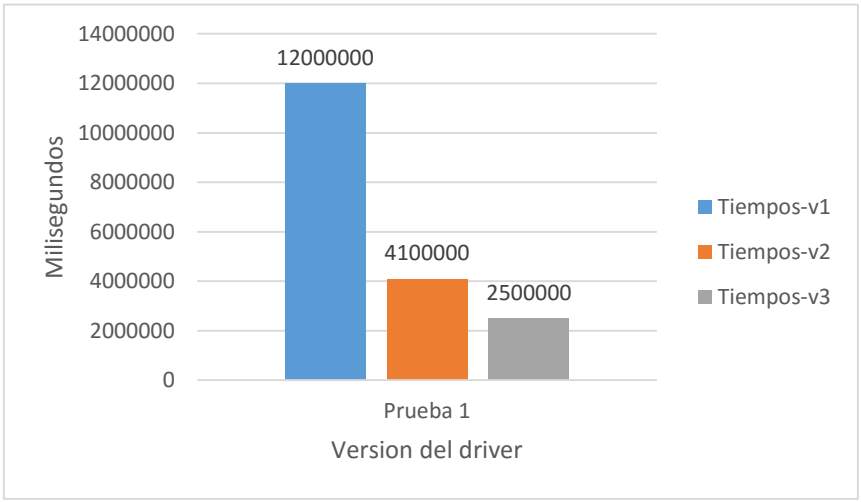
Por último tenemos la conclusión, donde se expondrán los principales problemas que hemos tenido a la hora de desarrollar la práctica y también las valoraciones de la misma.

2. Descripción del impacto de las mejoras.

¿Cansado de tener que esperar para poder acceder a los datos, imágenes, videos… que tienes guardados en tu disco duro? Ahora ofrecemos 3 versiones de un driver que te permitirá acceder a estos de manera más rápida y eficiente. A continuación mostramos como es de real esta mejora a través de una de las tantas pruebas que ha pasado nuestro driver.



Cada uno de los driver se adapta a las necesidades del usuario, pudiendo con el primero de ellos acceder a los datos de forma que no se pierda ningún dato y se facilitarán estos en el orden en el que se pidieron. En el segundo driver, para aquellos que requieran un mejor rendimiento, se aprovechan el lugar donde se almacenan los datos para que en media los datos se le faciliten de manera más rápida. Y por último, hemos diseñado la versión definitiva que aprovecha varios accesos que vayan a los mismos archivos y se tratan como uno solo, con lo que se consigue un extra de velocidad y por tanto de rendimiento, tal y como se muestran en las gráficas que mostramos más adelante.



Nuestro equipo de trabajo recomienda que se utilice la tercera versión del driver ya que se puede mejorar el rendimiento hasta 5 veces.

Los drivers son elementos de comunicación entre los distintos dispositivos con el ordenador, por lo que un buen diseño aumenta considerablemente la velocidad con la que estos responden. Ahora vamos a mostrar un ejemplo en el que nuestro driver para discos duros es considerablemente rápido y ha ido mejorando con cada una de las versiones.

3. Batería de pruebas.

A continuación vamos a mostrar las pruebas realizadas para comprobar el correcto funcionamiento del driver. En cada una de las pruebas vamos a analizar:

- Descripción de la prueba
- El objetivo de la prueba.
- Resumen del resultado obtenido

3.1. Para la versión 1 del driver.

3.1.1. Prueba 1.

Descripción de la prueba: Se ejecutará el código con un solo cliente que hace una lectura y escritura tanto al bloque 0 como al bloque 1.

Objetivo de la prueba: Comprobar que se realizan correctamente las escrituras y lecturas en distintos bloques del disco.

Resultado obtenido: Al tener un solo cliente tanto las lecturas como las escrituras se han realizado de manera secuencial y se obtiene y funciona correctamente el disco.

3.1.2. Prueba 2.

Descripción de la prueba: Se ejecutará el código con un solo cliente que realizara 2 escrituras consecutivas sobre el mismo bloque y una lectura posterior.

Objetivo de la prueba: Comprobar que se realizan correctamente las escrituras y en orden, y posteriormente en la lectura se recibe el valor de la segunda escritura.

Resultado obtenido: Se ha cumplido el objetivo de la prueba como se puede en la imagen.

```
BOOT: initializing the disk hardware...
BOOT: initializing the pkernel...
BOOT: creating the clients
CLIENT 2925: request update block 0
        DRIVER: request sent, pid: 2925, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: sending data, pid: 2925, block_id: 0
CLIENT 2925: block content after write request: 'a' ... 'a'
CLIENT 2925: request update block 0
        DRIVER: request sent, pid: 2925, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: sending data, pid: 2925, block_id: 0
CLIENT 2925: block content after write request: 'b' ... 'b'
CLIENT 2925: request read block 0
        DRIVER: request sent, pid: 2925, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 2925, block_id: 0
CLIENT 2925: block content after read request: 'b' ... 'b'
CLIENT 2925: END
BOOT: process 2925 returned 2925, status 256
                        DISK_HW: SIGINT received and now exiting

DISK_HW: Statistics:
DISK_HW: TOTAL_SEEK_TIME_MICROSEC 0
DISK_HW: TOTAL_TRANSFER_TIME_MICROSEC 300000
DISK_HW: TOTAL_SEEKS 0
DISK_HW: TOTAL_REQUESTS 3
PKERNEL: SIGINT received and now exiting
```

3.1.3. Prueba 3.

Descripción de la prueba: Se ejecutará el código con un solo cliente que realizara 2 escrituras consecutivas sobre distintos bloques y una lectura posterior del primero de los bloques escritos

Objetivo de la prueba: Comprobar que en la lectura se obtiene el valor del bloque solicitado, en el caso de esta prueba deberíamos obtener la del primer bloque.

Resultado obtenido: Como habíamos previsto, la lectura del primer bloque devuelve lo que se escribió en el al comienzo de la ejecución.

```
BOOT: initializing the disk hardware...
BOOT: initializing the pkernel...
BOOT: creating the clients
CLIENT 4449: request update block 0
        DRIVER: request sent, pid: 4449, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: sending data, pid: 4449, block_id: 0
CLIENT 4449: block content after write request: 'a' ... 'a'
CLIENT 4449: request update block 1
        DRIVER: request sent, pid: 4449, block_id: 1
                        DISK_HW: BLOCK_REQUESTED 1
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: sending data, pid: 4449, block_id: 1
CLIENT 4449: block content after write request: 'b' ... 'b'
CLIENT 4449: request update block 3
        DRIVER: request sent, pid: 4449, block_id: 3
                        DISK_HW: BLOCK_REQUESTED 3
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: sending data, pid: 4449, block_id: 3
CLIENT 4449: block content after write request: 'c' ... 'c'
CLIENT 4449: request read block 0
        DRIVER: request sent, pid: 4449, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 4449, block_id: 0
CLIENT 4449: block content after read request: 'a' ... 'a'
CLIENT 4449: END
BOOT: process 4449 returned 4449, status 256
PKERNEL: SIGINT received and now exiting
                        DISK_HW: SIGINT received and now exiting

DISK_HW: Statistics:
DISK_HW: TOTAL_SEEK_TIME_MICROSEC 300000
DISK_HW: TOTAL_TRANSFER_TIME_MICROSEC 400000
DISK_HW: TOTAL_SEEKS 6
DISK_HW: TOTAL_REQUESTS 4
```

3.2. Para la versión 2 del driver.

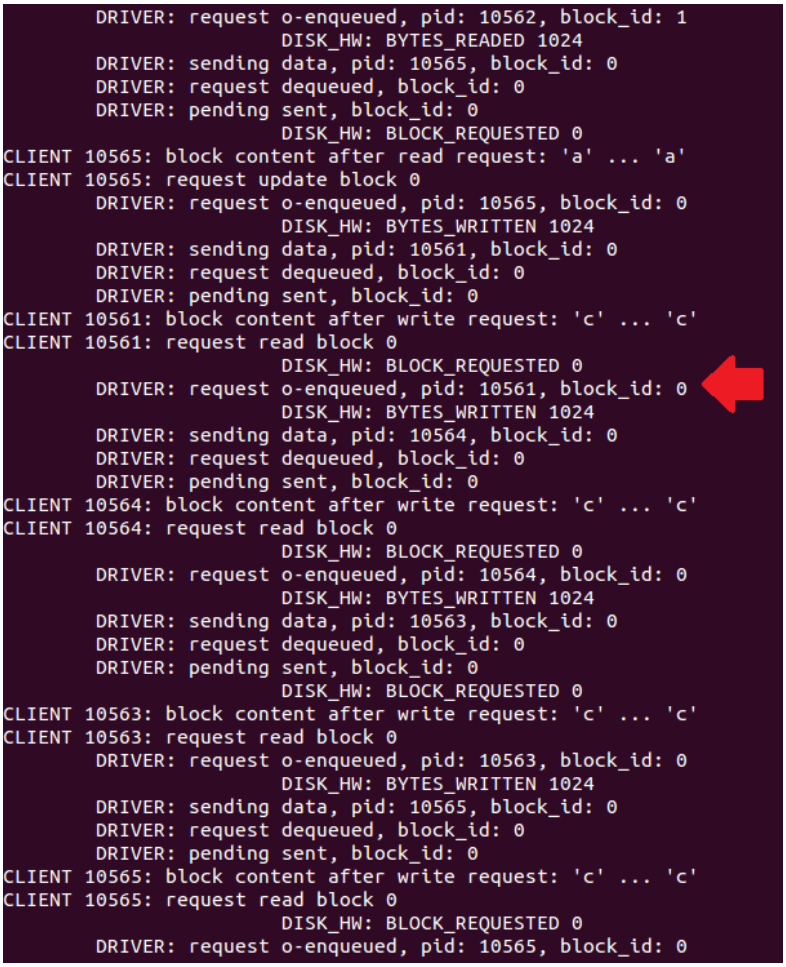
3.2.1. Prueba 1.

Descripción de la prueba: Vamos a lanzar la ejecución con 3 clientes. En este caso uno de los clientes, escribirá en el bloque 1 en un momento determinado y en otro momento (no a continuación) y luego los 3 leen el bloque 1 y continúan con el 0.

Objetivo de la prueba: Comprobar que las peticiones se ordenan en la cola en función del bloque que tratan.

Resultado obtenido: Parada 1: Cuando el cliente 1 intenta escribir en el bloque 1, se encola la petición después de las peticiones dirigidas al bloque 0. En cuanto el cliente 0 y 2 hace una petición para leer el bloque 1, se atiende la petición del cliente 1. La siguiente petición del cliente 1 es una lectura al bloque 0, por lo que se vuelve a bloquear hasta que 0 y 2 hayan leído el bloque

1. A continuación todos leen el bloque 0 hasta que el cliente 1 quiera realizar su lectura del bloque 1 y se encole su petición después de las del bloque 0. Cuando terminan el cliente 0 y 2el cliente 1 se ejecuta hasta terminar.



Como se puede ver en la imagen, desde donde señala la flecha no se ve ninguna ejecución del proceso 10562, ya que este intentado realizar una petición al bloque 1 mientras hay peticiones del bloque 0.

```
        DRIVER: request dequeued, block_id: 0
        DRIVER: pending sent, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
CLIENT 10563: block content after read request: 'c' ... 'c'
CLIENT 10563: END
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 10565, block_id: 0
        DRIVER: request dequeued, block_id: 1
        DRIVER: pending sent, block_id: 1
                        DISK_HW: BLOCK_REQUESTED 1
CLIENT 10565: block content after read request: 'c' ... 'c'
CLIENT 10565: END
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 10562, block_id: 1
CLIENT 10562: block content after read request: 'b' ... 'b'
CLIENT 10562: request read block 0
        DRIVER: request o-enqueued, pid: 10562, block_id: 0
        DRIVER: request sent, pid: 10562, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 10562, block_id: 0
CLIENT 10562: block content after read request: 'c' ... 'c'
CLIENT 10562: request read block 0
        DRIVER: request o-enqueued, pid: 10562, block_id: 0
        DRIVER: request sent, pid: 10562, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 10562, block_id: 0
CLIENT 10562: block content after read request: 'c' ... 'c'
CLIENT 10562: request read block 0
        DRIVER: request o-enqueued, pid: 10562, block_id: 0
        DRIVER: request sent, pid: 10562, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 10562, block_id: 0
CLIENT 10562: block content after read request: 'c' ... 'c'
CLIENT 10562: request read block 0
        DRIVER: request o-enqueued, pid: 10562, block_id: 0
        DRIVER: request sent, pid: 10562, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
```

En esta imagen, se puede ver como son atendidas las peticiones del cliente 10562. Esto se debe a que el resto de clientes han terminado ya sus peticiones mientras que este cliente hizo una petición al bloque 1 y se encolo detrás de las del bloque 0, por lo que hasta que no terminaron todas las peticiones del bloque 0, este no pudo obtener respuestas a sus peticiones.

3.3. Para la versión 3 del driver.

3.3.1. Prueba 1.

Descripción de la prueba: Vamos a lanzar la ejecución de 3 clientes con 15 peticiones de lectura cada uno.

Objetivo de la prueba: Comprobar que cuando se envía la primera lectura y a continuación vienen lecturas al mismo bloque estas deberán agruparse, de manera que solo se haga una única petición y que el resultado sea enviado a cada uno de los clientes que solicitaron las peticiones que se han agrupado.

Resultado obtenido: En las imágenes se puede observar cómo se agrupan peticiones, de manera que en total se hacen 13 cuando si no se hubieran agrupado se haría 45.

A continuación se muestran 2 imágenes que muestran el número total de peticiones realizadas al disco, comparado a las de la segunda versión del driver. También adjuntaremos una imagen que muestra cómo se agrupan las peticiones.

```
        DRIVER: request o-enqueued, pid: 9544, block_id: 0
        DRIVER: request sent, pid: 9544, block_id: 0
        DRIVER: request re-enqueued, pid: 9545, block_id: 0
        DRIVER: request re-enqueued, pid: 9543, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 9544, block_id: 0
        DRIVER: sending data, pid: 9545, block_id: 0
CLIENT 9545: block content after read request: 'c' ... 'c'
CLIENT 9545: request read block 0
CLIENT 9543: block content after read request: 'c' ... 'c'
CLIENT 9543: request read block 0
CLIENT 9544: block content after read request: 'c' ... 'c'
CLIENT 9544: request read block 0
        DRIVER: sending data, pid: 9543, block_id: 0
        DRIVER: request o-enqueued, pid: 9545, block_id: 0
        DRIVER: request sent, pid: 9545, block_id: 0
        DRIVER: request re-enqueued, pid: 9543, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
        DRIVER: request re-enqueued, pid: 9544, block_id: 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 9545, block_id: 0
        DRIVER: sending data, pid: 9543, block_id: 0
CLIENT 9545: block content after read request: 'c' ... 'c'
        DRIVER: sending data, pid: 9544, block_id: 0
CLIENT 9545: request read block 0
CLIENT 9543: block content after read request: 'c' ... 'c'
CLIENT 9543: request read block 0
CLIENT 9544: block content after read request: 'c' ... 'c'
CLIENT 9544: request read block 0
        DRIVER: request o-enqueued, pid: 9545, block_id: 0
        DRIVER: request sent, pid: 9545, block_id: 0
        DRIVER: request re-enqueued, pid: 9543, block_id: 0
        DRIVER: request re-enqueued, pid: 9544, block_id: 0
                        DISK_HW: BLOCK_REQUESTED 0
                        DISK_HW: BYTES_READED 1024
        DRIVER: sending data, pid: 9545, block_id: 0
        DRIVER: sending data, pid: 9543, block_id: 0
        DRIVER: sending data, pid: 9544, block_id: 0
CLIENT 9545: block content after read request: 'c' ... 'c'
CLIENT 9543: block content after read request: 'c' ... 'c'
```

4. Conclusiones.

Al inicio de la práctica tuvimos problemas a la hora de ejecutar ya que se nos llenaban las colas al terminar de forma abrupta. No supimos resolverlo hasta que Alejandro no mandó el correo con el comando que borraba las colas, con lo que perdimos tiempo intentando resolverlo por nuestra cuenta. Estaría bien que información de este tipo apareciese en el enunciado de la práctica para así evitar pérdidas de tiempo innecesarias.

Durante la realización de la práctica hemos tenido algunos problemas. En la parte 2 de la práctica, que trata sobre realizar una cola de peticiones ordenadas, tras haber escrito y revisado el código no encontrábamos el fallo que provocaba que no funcionara como debería. Tras varias horas perdidas nos dimos cuenta que el método de enqueue_ordered estaba mal implementado y por ello obteníamos los fallos durante la ejecución.

Para la resolución del tercer driver, implementamos un método para encolar peticiones que marcaba las peticiones de lectura dirigidas a un bloque que iba a ser modificado por una petición de escritura de manera que si había una nueva petición de lectura dirigida a ese bloque se crease una nueva petición y no se almacenase en las ya creadas. Invertimos bastante tiempo para la codificación de este método hasta que el viernes 1/4/2016 los profesores nos indicaron que se podía hacer de una manera más sencilla recorriendo la cola desde atrás.