PRÁCTICA 2: DRIVER DE DISPOSITIVOS DE BLOQUE

DISEÑO DE SISTEMAS OPERATIVOS

Curso 2015-2016

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

JAVIER PRIETO CEPEDA:	**100 307 011**
SERGIO RUIZ JIMÉNEZ:	**100 303 582**
MARIN LUCIAN PRIALA:	**100 303 625**

ÍNDICE

TABLAS DE PRUEBAS

1 Introducción

En este documento se procede a la explicación de la práctica 2, "Driver de dispositivos de driver", de la asignatura Diseño de Sistemas Operativos del grado de ingeniería informática de la Universidad Carlos III de Madrid. La realización de la misma la han llevado a cabo los alumnos Javier Prieto Cepeda, Marin Lucian Priala y Sergio Ruiz Jiménez.

En primer lugar se realizará una breve descripción del código implementado para los distintos drivers pedidos. Cabe destacar que son tres drivers distintos y cada uno de ellos está basado en el anterior con pequeñas modificaciones, que servirán para posteriormente analizar si son optimizaciones positivas.

Posteriormente, se compararán cada uno de los drivers, comprobando el impacto de las mejoras de cada uno.

En último lugar se comentarán las pruebas llevadas a cabo y los resultados de cada una de ellas, incluyendo al final del documento una breve conclusión de la práctica realizada.

2 Descripción del código

En el siguiente apartado, se procederá a explicar la implementación realizada para cada uno de los 3 drivers realizados, lo cuales han seguido una optimización incremental.

Cabe destacar, que todos los ficheros de los drivers son iguales, variando únicamente el valor de una constante, la cual determina la versión del driver. Su objetivo es guiar el flujo de la ejecución del código para ejecutar el driver deseado, evitando tener ficheros diferentes.

Para realizar éste fichero único, además, en el sistema de colas, se ha añadido una nueva función llamada "*find_request_impachetabil*", similar a "*find_request*", pero con la diferencia de que retorna un puntero al nodo de la lista en el cual insertar la nueva petición. La función ha sido implementada, fundamentalmente, para el segundo y tercer driver.

Además, ha sido necesaria la inclusión de mecanismos de concurrencia. En nuestro caso, hemos realizado la inclusión de mutex para asegurar que el tratamiento de las peticiones y las colas se hacen de manera exclusiva.

2.1 Encolar Peticiones

El primer driver implementado, ha recibido la optimización más sencilla. Esta optimización, se basa en guardar la peticiones de los clientes sobre el disco, de forma que cuando éste esté ocupado, las peticiones queden guardadas para ser procesadas una vez el disco vaya quedando disponible. Cada vez que una petición sea resuelta, será devuelta al cliente correspondiente.

De esta manera, cada vez que se recibe una petición de un cliente, ésta queda guardada en una cola (común a todos los clientes), y en caso de encontrarse libre el dispositivo, ésta será enviada al mismo para ser tratada.

Una vez está siendo tratada la petición, ésta se elimina de la cola. Cuando finaliza, se envía al cliente que tiene esa petición asociada. Una vez el dispositivo queda libre tras enviar la respuesta de la petición al cliente, este pide peticiones de cliente para tratarlas.

Para realizar ésta optimización, ha sido necesario modificar el archivo queue.h, incluyendo cambios en la estructura de request para poder almacenar correctamente las peticiones que llegan desde el usuario.

Como podemos ver este primer driver es una base para los siguientes, ya que no posee ninguna optimización importante, únicamente incluye una mejora respecto al driver proporcionado, de forma que no se pierdan peticiones.

2.2 Ordenamiento de peticiones

El segundo driver incluye una optimización que consiste en disminuir los tiempos de respuesta, insertando las peticiones de los clientes de forma ordenada por identificador de bloque. Con ello se consigue reducir el número de giros del cabezal hasta localizar el siguiente bloque, puesto que éste va de principio a fin tratando todas las peticiones encoladas.

La diferencia principal con respecto al primer driver, se encuentra en la inserción ordenada de las peticiones. De esta forma, si tenemos una nueva petición P_x, con un identificador de bloque X, quedará encolada tras la petición P_y, con un identificador de bloque Y<=X.

Cabe destacar que esta función fue modificada, debido a que la inserción ordenada entre dos elementos no estaba bien implementada. Con la corrección en la comparación a la hora de insertar en el límite inferior queda solucionado el error.

2.3 Agrupamiento de mismas peticiones

El tercer driver incluye una optimización considerable con respecto al driver anterior. Esta mejora, consiste en la agrupación o empaquetamiento de peticiones consecutivas que comparten un mismo identificador de bloque y operación, mientras que no exista entre ellas una petición al mismo identificador de bloque y distinta operación. Mediante esta técnica, se pretende reducir el número de peticiones enviadas a disco con el consecuente impacto en el rendimiento del sistema, disminuyendo tiempos globales en respuestas a peticiones.

El sistema de agrupación de peticiones, consiste en agrupamientos de hasta un máximo de 10 peticiones, de forma que cuando se trata un paquete de peticiones, sólo una es verdaderamente enviada al disco (aunque esto es invisible a los clientes), pudiendo reducir hasta en 9 veces el coste de peticiones.

Para ello, en el caso de las escrituras, únicamente se almacenará en el paquete, el bloque a escribir de la última petición. Además, en caso de ser la petición actual en disco, una escritura del mismo identificador de bloque que la petición recibida, y ser la petición recibida una escritura, pese a existir en el paquete de la petición en tratamiento espacio para la nueva, ésta se encolará como una nueva petición. Por el contrario, si fueran lecturas ambas, si se agruparían.

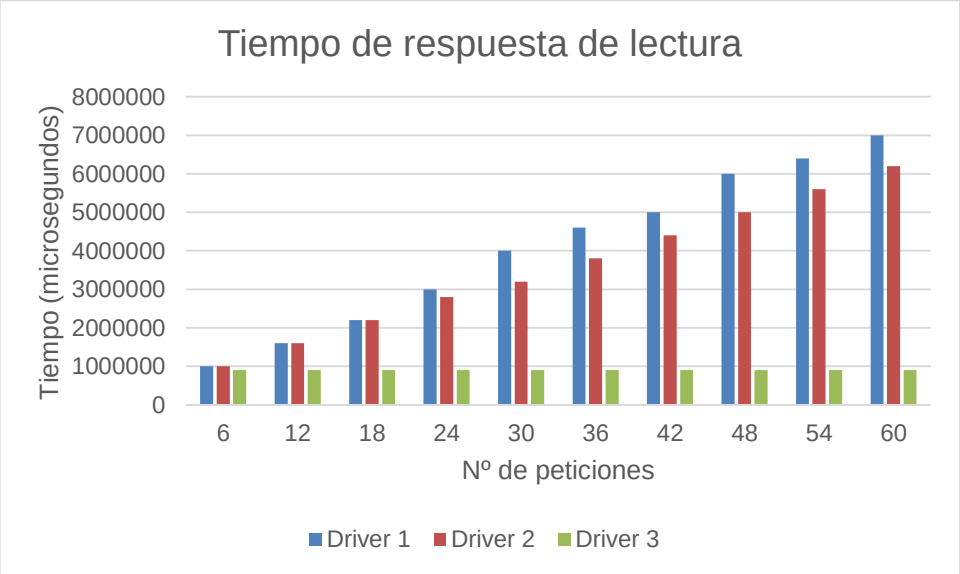
Es necesario comentar, que se podrían hacer agrupamientos de paquetes consecutivos y coincidentes en identificador de bloque y operación, de forma que cuando se envíe a disco la petición, se envíe únicamente la del último paquete, y una vez procesada por el dispositivo, se realice un backtracking para ir despertando a todos los procesos de las agrupaciones consecutivas coincidentes.

3 Impacto de las mejoras

En el siguiente apartado se mostrarán las estadísticas del disco para cada driver, mostrando así las mejoras de cada uno. Para ello citaremos los drivers como 1, 2 y 3. Siendo 1 el primer driver implementado y 3 el último driver.

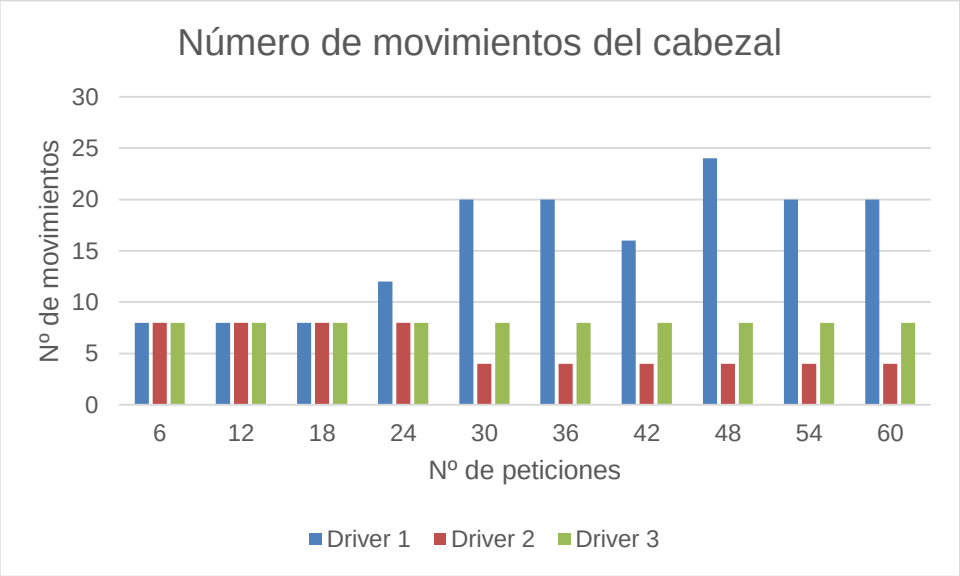
Las estadísticas tomadas dependen del número de peticiones realizadas, para ello se ha creado un cliente que genera 3 peticiones de lectura sobre bloques no consecutivos.

A continuación observamos el tiempo de respuesta de lectura de bloque para distinto número de peticiones, peor cuando el valor sea mayor. Podemos apreciar como para 6 peticiones el tiempo es muy parecido pero a medida que hay más peticiones, el driver número 3 mejora mucho los tiempos respecto a los otros dos drivers.



Gráfica 1. Tiempo de respuesta para operaciones de lectura

En la siguiente gráfica se aprecian los movimientos del cabeza, destacando el encolamiento en orden de las peticiones. Al encolar las peticiones dependiendo de su identificador de bloque, consigues proximidad al recorrer y tratar las peticiones, por lo tanto el cabezal deberá realizar menos movimientos que si estuvieran insertados sin orden (driver 1).



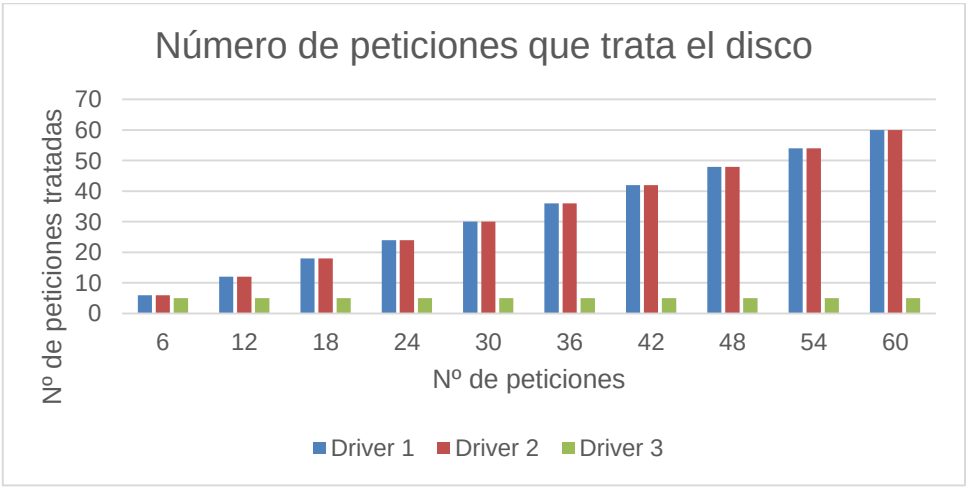
Gráfica 2. Número de movimientos del cabezal para operación de lectura

En esta gráfica es mucho más complejo apreciar la mejora entre el driver 2 y 3. Ya que ambos tienen la mejora de insertan en orden las peticiones.

Como se aprecia, hasta 18 peticiones el número de movimientos es similar. Aunque a partir de aquí comienzan las diferencias entre el driver 1 y el driver 2 y 3.

Puede parecer contradictorio ver mayor movimiento del cabezal para el driver 3, ya que los tiempos para el driver 3 son mejores que el driver 2, pero no lo es. Esto es debido al número de peticiones que realmente hace el disco, ya que si varios clientes reclaman el mismo bloque y operación, el disco no hará todas las peticiones pedidas. Tan solo hará 1 de ellas y devolverá el resultado a todos los clientes que lo han reclamado. Por lo tanto, el disco evita realizar muchas operaciones, reduciendo el tiempo de respuesta.

Para entender estos datos, también se ha realizado una gráfica con el número de peticiones que el disco trata respecto al número de peticiones que realmente se hacen por cliente, que siempre son 3.

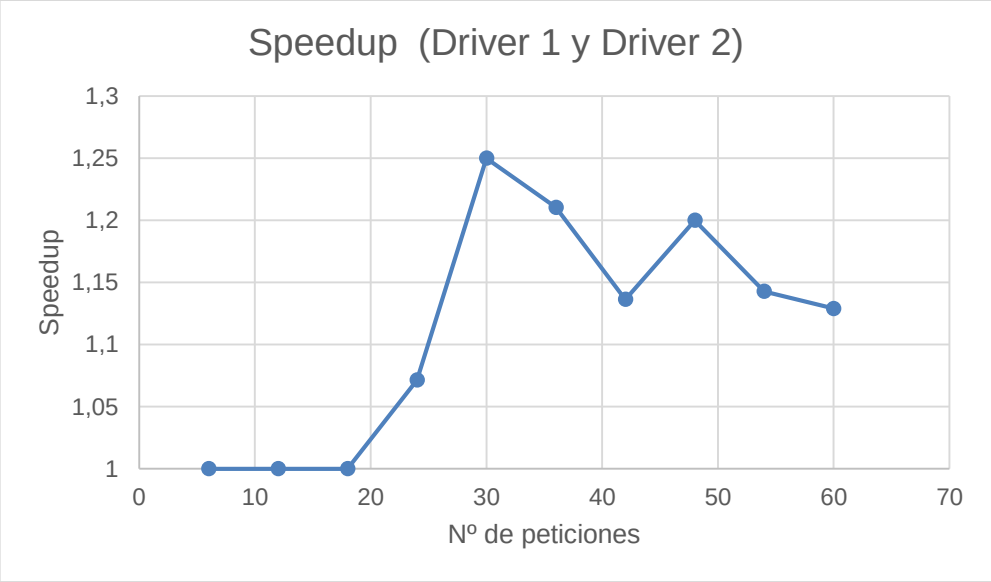


Gráfica 3. Número de peticiones realmente tratadas por el disco

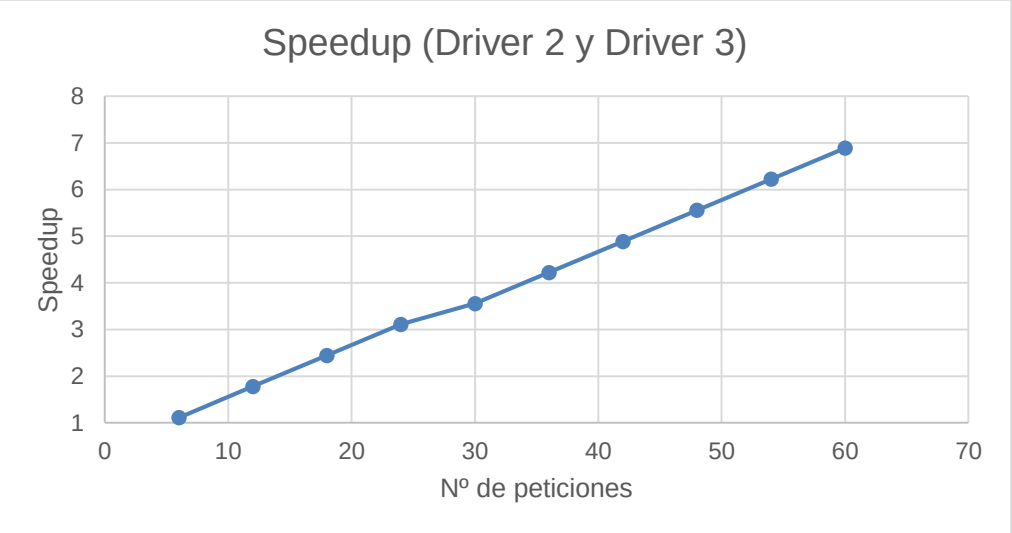
Esta gráfica es muy importante ya que está relacionada directamente con la velocidad de respuesta del disco. Como podemos ver, el driver 1 y 2 al no agrupar peticiones sobre mismo bloque y operación, siempre tienen que hacer todas las peticiones pedidas. A diferencia de ellos, el driver número 3, consigue agrupar estas peticiones si son similares para responder de una manera más rápida y eficiente a los clientes.

Esto quiere decir que nuestro driver número 3, consigue responder un 40% más rápido que el segundo mejor driver, el número 2. Cabe destacar también que el driver número 3 también reduce en un 40% las peticiones que trata el disco. Con ello se reduce la carga de trabajo del disco, consiguiendo una mayor duración de vida útil del dispositivo y una menor carga en la máquina.

A continuación, también se muestran unas gráficas del speedup de tiempo entre los drivers para apreciar las mejoras entre ellos.



Gráfica 4. Speedup tiempo entre driver 1 y 2



Gráfica 5. Speedup tiempo entre driver 2 y 3

4 Batería de pruebas

En este apartado, vamos a proceder a explicar las pruebas realizadas para el análisis del correcto funcionamiento de los 3 drivers. Cabe destacar, que las pruebas realizadas tratan de cubrir simultáneamente el máximo número de casos de prueba.

4.1 Encolar Peticiones

Para comprobar el correcto funcionamiento del encolado de peticiones, hemos realizado una simulación de ejecución, en la que 7 clientes diferentes, se encargan de realizar peticiones al disco.

Resultado Esperado: De esta forma, la primera petición en llegar, debe de encolarse, y enviarse el dispositivo. Hasta que el dispositivo quede libre, el resto de peticiones deben de irse encolando, de forma que se vayan mandando al dispositivo en orden de llegada (siguiendo una política FIFO).

```
BOOT: initializing the disk hardware...
BOOT: initializing the pkernel...
BOOT: creating the clients
CLIENT 26629: request read block 5
        DRIVER: request enqueued, pid: 26629, block_id: 5
        DRIVER: request sent, pid: 26629, block_id: 5
                        DISK_HW: BLOCK_REQUESTED 5
CLIENT 26633: request update block 15
CLIENT 26635: request update block 15
CLIENT 26631: request update block 15
        DRIVER: request enqueued, pid: 26633, block_id: 15
        DRIVER: request enqueued, pid: 26635, block_id: 15
        DRIVER: request enqueued, pid: 26631, block_id: 15
CLIENT 26634: request read block 6
CLIENT 26630: request read block 6
CLIENT 26632: request read block 6
        DRIVER: request enqueued, pid: 26634, block_id: 6
        DRIVER: request enqueued, pid: 26630, block_id: 6
        DRIVER: request enqueued, pid: 26632, block_id: 6
                        DISK_HW: BYTES_READED 1024
        DRIVER: request dequeued, block_id: 5
        DRIVER: sending data, pid: 26629, block_id: 5
        DRIVER: pending sent, block_id: 15
CLIENT 26629: block content after read request: '0' ... '0'
                        DISK_HW: BLOCK_REQUESTED 15
CLIENT 26629: request update block 15
        DRIVER: request enqueued, pid: 26629, block_id: 15
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26633, block_id: 15
        DRIVER: pending sent, block_id: 15
CLIENT 26633: block content after write request: 'b' ... 'b'
                        DISK_HW: BLOCK_REQUESTED 15
CLIENT 26633: request update block 15
        DRIVER: request enqueued, pid: 26633, block_id: 15
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26635, block_id: 15
        DRIVER: pending sent, block_id: 15
CLIENT 26635: block content after write request: 'b' ... 'b'
                        DISK_HW: BLOCK_REQUESTED 15
```

```
CLIENT 26635: request update block 15
        DRIVER: request enqueued, pid: 26635, block_id: 15
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26631, block_id: 15
        DRIVER: pending sent, block_id: 6
CLIENT 26631: block content after write request: 'b' ... 'b'
                        DISK_HW: BLOCK_REQUESTED 6
CLIENT 26631: request update block 15
        DRIVER: request enqueued, pid: 26631, block_id: 15
                        DISK_HW: BYTES_READED 1024
        DRIVER: request dequeued, block_id: 6
        DRIVER: sending data, pid: 26634, block_id: 6
        DRIVER: pending sent, block_id: 6
CLIENT 26634: block content after read request: 'b' ... 'b'
CLIENT 26634: END
                        DISK_HW: BLOCK_REQUESTED 6
                        DISK_HW: BYTES_READED 1024
        DRIVER: request dequeued, block_id: 6
        DRIVER: sending data, pid: 26630, block_id: 6
        DRIVER: pending sent, block_id: 6
CLIENT 26630: block content after read request: 'b' ... 'b'
                        DISK_HW: BLOCK_REQUESTED 6
CLIENT 26630: END
                        DISK_HW: BYTES_READED 1024
        DRIVER: request dequeued, block_id: 6
        DRIVER: sending data, pid: 26632, block_id: 6
        DRIVER: pending sent, block_id: 15
                        DISK_HW: BLOCK_REQUESTED 15
CLIENT 26632: block content after read request: 'b' ... 'b'
CLIENT 26632: END
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26629, block_id: 15
        DRIVER: pending sent, block_id: 15
CLIENT 26629: block content after write request: '0' ... '0'
                        DISK_HW: BLOCK_REQUESTED 15
CLIENT 26629: request update block 15
        DRIVER: request enqueued, pid: 26629, block_id: 15
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26633, block_id: 15
        DRIVER: pending sent, block_id: 15
CLIENT 26633: block content after write request: 'b' ... 'b'
                        DISK_HW: BLOCK_REQUESTED 15
CLIENT 26633: END
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26635, block_id: 15
        DRIVER: pending sent, block_id: 15
CLIENT 26635: block content after write request: 'b' ... 'b'
CLIENT 26635: END
                        DISK_HW: BLOCK_REQUESTED 15
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26631, block_id: 15
        DRIVER: pending sent, block_id: 15
CLIENT 26631: block content after write request: 'b' ... 'b'
CLIENT 26631: END
                        DISK_HW: BLOCK_REQUESTED 15
```

```
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 26629, block_id: 15
CLIENT 26629: block content after write request: '0' ... '0'
CLIENT 26629: END
BOOT: process 26629 returned 26629, status 256
BOOT: process 26630 returned 26630, status 256
BOOT: process 26631 returned 26631, status 256
BOOT: process 26632 returned 26632, status 256
BOOT: process 26633 returned 26633, status 256
BOOT: process 26634 returned 26634, status 256
BOOT: process 26635 returned 26635, status 256
                        DISK_HW: SIGINT received and now exiting

DISK_HW: Statistics:
DISK_HW: TOTAL_SEEK_TIME_MICROSEC 16500000
DISK_HW: TOTAL_TRANSFER_TIME_MICROSEC 12000000
DISK_HW: TOTAL_SEEKS 33
DISK_HW: TOTAL_REQUESTS 12
PKERNEL: SIGINT received and now exiting
```

Tabla 1: Traza Prueba Encolado

Resultado Obtenido: Como podemos observar, cuando llega la primera petición, esta se encola y es mandado al disco, puesto que este está libre. Mientras el disco está ocupado, todas las nuevas peticiones, van siendo encoladas, de forma que cuando el disco finaliza con la primera, toma la primera petición de la cola (la primera que llegó). De este modo, queda verificado el correcto funcionamiento del enconamiento de peticiones.

4.2 Ordenamiento de peticiones

Para comprobar el correcto funcionamiento del encolado ordenado de peticiones, hemos realizado una simulación de ejecución, en la que 7 clientes diferentes, se encargan de realizar peticiones al disco.

Resultado Esperado: De esta forma, la primera petición en llegar, la cual será una lectura del bloque 5, será encolada, y posteriormente enviada al dispositivo. Después, se realizarán una serie de peticiones de escritura del bloque 15, y tras ellas, 3 peticiones de lectura del bloque 6. Al ser una inserción ordenada en este driver, una vez finalice el dispositivo de tratar la lectura del bloque 5, debería de realizar la lectura del bloque 6, al ser la más cercana, en lugar de realizar la escritura del bloque 15, que llegó antes.

```
BOOT: initializing the disk hardware...
BOOT: initializing the pkernel...
BOOT: creating the clients
CLIENT 24876: request read block 5
        DRIVER: request o-enqueued, pid: 24876, block_id: 5
        DRIVER: request sent, pid: 24876, block_id: 5
                        DISK_HW: BLOCK_REQUESTED 5
CLIENT 24878: request update block 15
CLIENT 24880: request update block 15
CLIENT 24882: request update block 15
        DRIVER: request o-enqueued, pid: 24878, block_id: 15
        DRIVER: request o-enqueued, pid: 24880, block_id: 15
        DRIVER: request o-enqueued, pid: 24882, block_id: 15
CLIENT 24881: request read block 6
CLIENT 24879: request read block 6
CLIENT 24877: request read block 6
        DRIVER: request o-enqueued, pid: 24881, block_id: 6
        DRIVER: request o-enqueued, pid: 24879, block_id: 6
        DRIVER: request o-enqueued, pid: 24877, block_id: 6
                        DISK_HW: BYTES_READED 1024
        DRIVER: request dequeued, block_id: 5
        DRIVER: sending data, pid: 24876, block_id: 5
        DRIVER: pending sent, block_id: 6
CLIENT 24876: block content after read request: '0' ... '0'
                        DISK_HW: BLOCK_REQUESTED 6
CLIENT 24876: request update block 15
        DRIVER: request o-enqueued, pid: 24876, block_id: 15
                        DISK_HW: BYTES_READED 1024
        DRIVER: request dequeued, block_id: 6
        DRIVER: sending data, pid: 24881, block_id: 6
        DRIVER: request dequeued, block_id: 6
        DRIVER: sending data, pid: 24879, block_id: 6
        DRIVER: request dequeued, block_id: 6
CLIENT 24881: block content after read request: 'b' ... 'b'
        DRIVER: sending data, pid: 24877, block_id: 6
CLIENT 24879: block content after read request: 'b' ... 'b'
CLIENT 24881: END
        DRIVER: pending sent, block_id: 15
                        DISK_HW: BLOCK_REQUESTED 15
CLIENT 24879: END
CLIENT 24877: block content after read request: 'b' ... 'b'
CLIENT 24877: END
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 24878, block_id: 15
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 24880, block_id: 15
CLIENT 24878: block content after write request: 'b' ... 'b'
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 24882, block_id: 15
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 24876, block_id: 15
CLIENT 24882: block content after write request: 'b' ... 'b'
CLIENT 24880: block content after write request: 'b' ... 'b'
CLIENT 24876: block content after write request: 'b' ... 'b'
CLIENT 24878: request update block 15
CLIENT 24876: request update block 15
CLIENT 24882: request update block 15
CLIENT 24880: request update block 15
        DRIVER: request o-enqueued, pid: 24878, block_id: 15
```

```
        DRIVER: request sent, pid: 24878, block_id: 15
        DRIVER: request o-enqueued, pid: 24876, block_id: 15
        DRIVER: request o-enqueued, pid: 24882, block_id: 15
        DRIVER: request o-enqueued, pid: 24880, block_id: 15
                        DISK_HW: BLOCK_REQUESTED 15
                        DISK_HW: BYTES_WRITTEN 1024
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 24878, block_id: 15
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 24876, block_id: 15
        DRIVER: request dequeued, block_id: 15
CLIENT 24876: block content after write request: 'b' ... 'b'
        DRIVER: sending data, pid: 24882, block_id: 15
        DRIVER: request dequeued, block_id: 15
CLIENT 24882: block content after write request: 'b' ... 'b'
CLIENT 24878: block content after write request: 'b' ... 'b'
CLIENT 24876: END
        DRIVER: sending data, pid: 24880, block_id: 15
CLIENT 24882: END
CLIENT 24880: block content after write request: 'b' ... 'b'
CLIENT 24878: END
CLIENT 24880: END
BOOT: process 24876 returned 24876, status 256
BOOT: process 24877 returned 24877, status 256
BOOT: process 24878 returned 24878, status 256
BOOT: process 24879 returned 24879, status 256
BOOT: process 24880 returned 24880, status 256
BOOT: process 24881 returned 24881, status 256
BOOT: process 24882 returned 24882, status 256
                        DISK_HW: SIGINT received and now exiting
PKERNEL: SIGINT received and now exiting

DISK_HW: Statistics:
DISK_HW: TOTAL_SEEK_TIME_MICROSEC 7500000
DISK_HW: TOTAL_TRANSFER_TIME_MICROSEC 4000000
DISK_HW: TOTAL_SEEKS 15
DISK_HW: TOTAL_REQUESTS 4
```

Tabla 2: Traza Prueba Ordenación

Resultado Obtenido: Como podemos observar, en primer lugar, se recibe la petición de lectura del bloque 5. Tras él, se producen 3 peticiones de escritura del bloque 15 y 3 peticiones de lectura del bloque 6. Una vez finaliza el dispositivo de realizar la primera petición, desencola la primera petición de lectura del bloque 6, puesto que es de las más cercanas, la primera en llegar. De este modo, queda verificado el correcto funcionamiento de la ordenación de peticiones.

4.3 Agrupamiento de mismas peticiones

En esta ocasión, para comprobar el correcto funcionamiento del tercer driver, hemos decidido realizar dos pruebas diferentes. En la primera, buscamos verificar el correcto funcionamiento del driver, de forma que se encolen de forma ordenada las peticiones (como en el driver 2 quedó demostrado) pero que además, se demuestre el correcto empaquetamiento de las peticiones. En la segunda prueba, buscamos verificar que una vez se llena un paquete, la siguiente petición que llegue y podría llegar a ser empaquetada en caso de haber espacio en el paquete, no se empaqueta, y se crea una nueva petición.

4.3.1 Agrupamiento ordenado 1

Resultado Esperado: En esta prueba, pretendemos comprobar que se empaquetan las peticiones de forma correcta. En primer lugar, llegará una lectura del bloque 5, que se encolará y se enviará al dispositivo. Mientras este se encuentra ocupado, llegarán 3 peticiones de escritura al bloque 5, de las cuales la primera creará una nueva petición, y las dos posteriores, se empaquetarán a ésta. Posteriormente, llegarán 3 peticiones de lectura del bloque 5. La primera crea una nueva petición y las posteriores, se pueden empaquetar a ella. Cuando finalice la petición actual, deberá de enviarse al dispositivo la última petición de escritura (el agrupamiento) y despertar a todos los procesos del paquete de lectura del bloque 5 en tratamiento.

```
BOOT: initializing the disk hardware...
BOOT: initializing the pkernel...
BOOT: creating the clients
CLIENT 4261: request read block 5
      DRIVER: request o-enqueued, pid: 4261, block_id: 5
      DRIVER: request sent, pid: 4261, block_id: 5
                  DISK_HW: BLOCK_REQUESTED 5
CLIENT 4265: request update block 5
CLIENT 4267: request update block 5
CLIENT 4263: request update block 5
      DRIVER: request o-enqueued, pid: 4265, block_id: 5
      DRIVER: request re-enqueued, pid: 4267, block_id: 5
      DRIVER: request re-enqueued, pid: 4263, block_id: 5
CLIENT 4264: request read block 5
      DRIVER: request o-enqueued, pid: 4264, block_id: 5
CLIENT 4266: request read block 5
CLIENT 4262: request read block 5
      DRIVER: request re-enqueued, pid: 4262, block_id: 5
      DRIVER: request re-enqueued, pid: 4266, block_id: 5
                  DISK_HW: BYTES_READED 1024
      DRIVER: request dequeued, block_id: 5
      DRIVER: sending data, pid: 4261, block_id: 5
      DRIVER: pending sent, block_id: 5
CLIENT 4261: block content after read request: '0' ... '0'
                  DISK_HW: BLOCK_REQUESTED 5
CLIENT 4261: request update block 5
      DRIVER: request o-enqueued, pid: 4261, block_id: 5
                  DISK_HW: BYTES_WRITTEN 1024
      DRIVER: request dequeued, block_id: 5
      DRIVER: sending data, pid: 4265, block_id: 5
CLIENT 4265: block content after write request: 'b' ... 'b'
CLIENT 4267: block content after write request: 'b' ... 'b'
      DRIVER: sending data, pid: 4267, block_id: 5
CLIENT 4263: block content after write request: 'b' ... 'b'
      DRIVER: sending data, pid: 4263, block_id: 5
                  DISK_HW: BLOCK_REQUESTED 5
      DRIVER: pending sent, block_id: 5
CLIENT 4265: request update block 15
CLIENT 4263: request update block 15
CLIENT 4267: request update block 15
      DRIVER: request o-enqueued, pid: 4265, block_id: 15
      DRIVER: request re-enqueued, pid: 4263, block_id: 15
      DRIVER: request re-enqueued, pid: 4267, block_id: 15
                  DISK_HW: BYTES_READED 1024
```

```
      DRIVER: request dequeued, block_id: 5
      DRIVER: sending data, pid: 4264, block_id: 5
CLIENT 4264: block content after read request: 'b' ... 'b'
CLIENT 4264: END
CLIENT 4262: block content after read request: 'b' ... 'b'
CLIENT 4262: END
      DRIVER: sending data, pid: 4262, block_id: 5
CLIENT 4266: block content after read request: 'b' ... 'b'
CLIENT 4266: END
      DRIVER: sending data, pid: 4266, block_id: 5
                  DISK_HW: BLOCK_REQUESTED 5
      DRIVER: pending sent, block_id: 5
                  DISK_HW: BYTES_WRITTEN 1024
      DRIVER: request dequeued, block_id: 5
CLIENT 4261: block content after write request: '0' ... '0'
      DRIVER: sending data, pid: 4261, block_id: 5
      DRIVER: pending sent, block_id: 15
                  DISK_HW: BLOCK_REQUESTED 15
CLIENT 4261: request update block 15
      DRIVER: request o-enqueued, pid: 4261, block_id: 15
                  DISK_HW: BYTES_WRITTEN 1024
      DRIVER: request dequeued, block_id: 15
CLIENT 4265: block content after write request: 'b' ... 'b'
CLIENT 4265: END
      DRIVER: sending data, pid: 4265, block_id: 15
CLIENT 4263: block content after write request: 'b' ... 'b'
CLIENT 4263: END
      DRIVER: sending data, pid: 4263, block_id: 15
CLIENT 4267: block content after write request: 'b' ... 'b'
CLIENT 4267: END
      DRIVER: sending data, pid: 4267, block_id: 15
      DRIVER: request dequeued, block_id: 15
CLIENT 4261: block content after write request: 'b' ... 'b'
CLIENT 4261: END
BOOT: process 4261 returned 4261, status 256
BOOT: process 4262 returned 4262, status 256
BOOT: process 4263 returned 4263, status 256
BOOT: process 4264 returned 4264, status 256
BOOT: process 4265 returned 4265, status 256
BOOT: process 4266 returned 4266, status 256
BOOT: process 4267 returned 4267, status 256
                  DISK_HW: SIGINT received and now exiting

DISK_HW: Statistics:
DISK_HW: TOTAL_SEEK_TIME_MICROSEC 750000
DISK_HW: TOTAL_TRANSFER_TIME_MICROSEC 500000
DISK_HW: TOTAL_SEEKS 15
DISK_HW: TOTAL_REQUESTS 5
PKERNEL: SIGINT received and now exiting
```

Tabla 3: Traza Prueba Agrupamiento Ordenado1

Resultado Obtenido: Resultado Obtenido: Como podemos observar, llega la primera petición de lectura del bloque 5. Esta se encola, y se envía al dispositivo. Posteriormente, llegan una serie de peticiones de escritura del bloque 5, de forma que la primera creará una nueva petición, y las posteriores se empaquetan a esta. Cuando se finaliza la lectura del bloque 5, se envía a disco la escritura del bloque 5,

enviándose la última escritura del grupo. Por tanto, podemos concluir que se realiza correctamente.

4.3.2 Agrupamiento ordenado 2

Resultado Esperado: En esta prueba, pretendemos comprobar que se empaquetan las peticiones de forma correcta. En primer lugar, llegará una lectura del bloque 5, que se encolará y se enviará al dispositivo. Mientras este se encuentra ocupado, llegará escritura del bloque 5, y 2 lecturas del bloque 6. Las escrituras del bloque 5, generarán una nueva petición. Las 2 lecturas del bloque 6, generarán una nueva petición, empaquetándose la segunda. Posteriormente, llegarán una escritura del bloque 5 y una lectura del bloque 6, que deberán de agruparse en sus respectivos grupos.

```
BOOT: initializing the disk hardware...
BOOT: initializing the pkernel...
BOOT: creating the clients
CLIENT 4403: request read block 5
      DRIVER: request o-enqueued, pid: 4403, block_id: 5
      DRIVER: request sent, pid: 4403, block_id: 5
                  DISK_HW: BLOCK_REQUESTED 5
CLIENT 4407: request update block 5
CLIENT 4406: request read block 6
CLIENT 4408: request read block 6
CLIENT 4405: request update block 5
CLIENT 4404: request read block 6
CLIENT 4409: request update block 5
      DRIVER: request o-enqueued, pid: 4407, block_id: 5
      DRIVER: request o-enqueued, pid: 4406, block_id: 6
      DRIVER: request re-enqueued, pid: 4408, block_id: 6
      DRIVER: request re-enqueued, pid: 4405, block_id: 5
      DRIVER: request re-enqueued, pid: 4404, block_id: 6
      DRIVER: request re-enqueued, pid: 4409, block_id: 5
                  DISK_HW: BYTES_READED 1024
      DRIVER: request dequeued, block_id: 5
CLIENT 4403: block content after read request: 'b' ... 'b'
      DRIVER: sending data, pid: 4403, block_id: 5
      DRIVER: pending sent, block_id: 5
                  DISK_HW: BLOCK_REQUESTED 5
CLIENT 4403: request update block 5
      DRIVER: request o-enqueued, pid: 4403, block_id: 5
                  DISK_HW: BYTES_WRITTEN 1024
      DRIVER: request dequeued, block_id: 5
      DRIVER: sending data, pid: 4407, block_id: 5
CLIENT 4405: block content after write request: 'b' ... 'b'
      DRIVER: sending data, pid: 4405, block_id: 5
CLIENT 4409: block content after write request: 'b' ... 'b'
      DRIVER: sending data, pid: 4409, block_id: 5
      DRIVER: request dequeued, block_id: 5
CLIENT 4407: block content after write request: 'b' ... 'b'
CLIENT 4403: block content after write request: 'b' ... 'b'
      DRIVER: sending data, pid: 4403, block_id: 5
      DRIVER: pending sent, block_id: 6
                  DISK_HW: BLOCK_REQUESTED 6
CLIENT 4405: request update block 15
CLIENT 4409: request update block 15
CLIENT 4403: request update block 15
```

```
      DRIVER: request o-enqueued, pid: 4405, block_id: 15
      DRIVER: request re-enqueued, pid: 4409, block_id: 15
      DRIVER: request re-enqueued, pid: 4403, block_id: 15
CLIENT 4407: request update block 15
      DRIVER: request re-enqueued, pid: 4407, block_id: 15
                  DISK_HW: BYTES_READED 1024
      DRIVER: request dequeued, block_id: 6
CLIENT 4406: block content after read request: '0' ... '0'
CLIENT 4406: END
      DRIVER: sending data, pid: 4406, block_id: 6
CLIENT 4408: block content after read request: '0' ... '0'
CLIENT 4408: END
      DRIVER: sending data, pid: 4408, block_id: 6
CLIENT 4404: block content after read request: '0' ... '0'
CLIENT 4404: END
      DRIVER: sending data, pid: 4404, block_id: 6
      DRIVER: pending sent, block_id: 15
                  DISK_HW: BLOCK_REQUESTED 15
                  DISK_HW: BYTES_WRITTEN 1024
      DRIVER: request dequeued, block_id: 15
      DRIVER: sending data, pid: 4405, block_id: 15
      DRIVER: sending data, pid: 4409, block_id: 15
      DRIVER: sending data, pid: 4403, block_id: 15
CLIENT 4405: block content after write request: 'b' ... 'b'
      DRIVER: sending data, pid: 4407, block_id: 15
CLIENT 4405: END
CLIENT 4409: block content after write request: 'b' ... 'b'
CLIENT 4409: END
CLIENT 4403: block content after write request: 'b' ... 'b'
CLIENT 4403: END
CLIENT 4407: block content after write request: 'b' ... 'b'
CLIENT 4407: END
BOOT: process 4403 returned 4403, status 256
BOOT: process 4404 returned 4404, status 256
BOOT: process 4405 returned 4405, status 256
BOOT: process 4406 returned 4406, status 256
BOOT: process 4407 returned 4407, status 256
BOOT: process 4408 returned 4408, status 256
BOOT: process 4409 returned 4409, status 256
PKERNEL: SIGINT received and now exiting
                  DISK_HW: SIGINT received and now exiting

DISK_HW: Statistics:
DISK_HW: TOTAL_SEEK_TIME_MICROSEC 750000
DISK_HW: TOTAL_TRANSFER_TIME_MICROSEC 400000
DISK_HW: TOTAL_SEEKS 15
DISK_HW: TOTAL_REQUESTS 4
```

Tabla 4: Traza Prueba Agrupamiento Ordenado 2

Resultado Obtenido: Como podemos observar, llega la primera petición de lectura del bloque 5, que se encola, y se envía al dispositivo. Posteriormente, llega una escritura del bloque 5, creando una nueva petición y encolándose. Luego llegan 2 peticiones de lectura del bloque 6, creándose una nueva petición, y encolándose la segunda. Luego llega una escritura del bloque 5, que se empaqueta, y posteriormente una lectura del bloque 6, que se empaqueta. Por tanto, podemos verificar, que se ejecuta correctamente.

4.3.3 Llenado de grupo de peticiones

Resultado Esperado: En esta prueba, pretendemos comprobar que cuando se llena un paquete de 10 peticiones, al llegar una igual, y no poder empaquetarse, se crea una nueva petición y se encola. Para ello, lanzamos 1 lectura del bloque 5, y 11 escrituras del bloque 15. La primera lectura, deberá crear una nueva petición, y ser enviada al dispositivo. La primera escritura recibida, deberá generar una nueva petición, y las 9 escrituras posteriores, deberán empaquetarse a ésta. La última escritura, al no entrar en el paquete, deberá generar una nueva petición, y ser encolada.

```
DRIVER: request o-enqueued, pid: 3140, block_id: 5
        DRIVER: request sent, pid: 3140, block_id: 5
        DRIVER: request o-enqueued, pid: 3143, block_id: 15
        DRIVER: request re-enqueued, pid: 3149, block_id: 15
        DRIVER: request re-enqueued, pid: 3145, block_id: 15
        DRIVER: request re-enqueued, pid: 3147, block_id: 15
        DRIVER: request re-enqueued, pid: 3151, block_id: 15
        DRIVER: request re-enqueued, pid: 3144, block_id: 15
        DRIVER: request re-enqueued, pid: 3150, block_id: 15
        DRIVER: request re-enqueued, pid: 3146, block_id: 15
        DRIVER: request re-enqueued, pid: 3141, block_id: 15
        DRIVER: request re-enqueued, pid: 3142, block_id: 15
        DRIVER: request o-enqueued, pid: 3148, block_id: 15
        DRIVER: request dequeued, block_id: 5
        DRIVER: sending data, pid: 3140, block_id: 5
        DRIVER: pending sent, block_id: 15
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 3143, block_id: 15
        DRIVER: sending data, pid: 3149, block_id: 15
        DRIVER: sending data, pid: 3145, block_id: 15
        DRIVER: sending data, pid: 3147, block_id: 15
        DRIVER: sending data, pid: 3151, block_id: 15
        DRIVER: sending data, pid: 3144, block_id: 15
        DRIVER: sending data, pid: 3150, block_id: 15
        DRIVER: sending data, pid: 3146, block_id: 15
        DRIVER: sending data, pid: 3141, block_id: 15
        DRIVER: sending data, pid: 3142, block_id: 15
        DRIVER: request dequeued, block_id: 15
        DRIVER: sending data, pid: 3148, block_id: 15
PKERNEL: SIGINT received and now exiting
```

Tabla 5: Traza Llenado Grupo Peticiones

Resultado Obtenido: Como podemos observar, llega la primera petición de lectura, que es encolada, y enviada al dispositivo. Posteriormente, llega la primera escritura, que crea una nueva petición, y es encolada. Después, las 9 escrituras posteriores, se empaquetan a ésta. Cuando el paquete de 10 se encuentra completo y llega la última escritura, se crea una nueva petición al no entrar en el paquete, y se encola. Por tanto, podemos verificar, que el comportamiento es el correcto.

5 Conclusiones

Como se ha podido apreciar a lo largo del documento, las implementaciones de la práctica han sido incrementales, de forma que se han ido añadiendo optimizaciones progresivas sobre los drivers anteriores, obteniendo un driver más complejo a la vez que optimizado que el primero, que haga de nexo entre una simulación de kernel sencillo y un dispositivo de bloques ficticio.

Se han ido observando las mejoras de cada driver desde el principio, ya que al principio no era capaz de guardar las peticiones del cliente en caso de estar ocupado el disco. Posteriormente fue capaz de guardarlas sin ninguna mejora más, para después insertar peticiones de manera ordenada reduciendo el número de veces que el cabezal gira para encontrar el bloque requerido. Y por último como se agrupan y ordenan las peticiones según su tipo para reducir el número real de peticiones que trata el disco.

Cabe destacar, que al principio de la implementación se hicieron drivers mejorados desde un principio. Por ejemplo, en el driver número dos, se consiguió sin crear una estructura auxiliar donde almacenar peticiones iguales, mejorar el tiempo de respuesta. Esto fue mediante una comprobación en la cola principal, en la que se observaba si había una escritura y posteriormente otra escritura sobre el mismo bloque para únicamente realizar una escritura sobre el bloque y reducir las peticiones que el disco realmente hace. Esto fue sustituido por el driver que se pedía explícitamente, para evitar problemas en la corrección.

También cabe destacar, que pese a indicar al final de la descripción del tercer driver una posible mejora, ésta no ha sido llevada a cabo por la falta de tiempo y no ceñirse a los requisitos del enunciado de la práctica.

Además en el segundo driver, podría obtenerse una mejora considerable si tras llegar al final de las peticiones, en la vuelta al comienzo se van recogiendo las posibles peticiones encoladas hasta que se llegue al comienzo del disco, de forma que siempre se traten las peticiones más cercanas, haciendo un funcionamiento parecido al de un ascensor (0-->n ; n-->0).

En definitiva, realizar un análisis e implementación de un driver como éste, es muy útil para poder comprender el funcionamiento entre el sistema operativo y un dispositivo y ver cómo influye en el rendimiento las distintas operaciones que se realizan sobre éstos.