PRÁCTICA 2: E/S ASÍNCRONA EN C

DISEÑO DE SOFTWARE DE SISTEMAS

Curso 2015-2016

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

ALONSO MONSALVE, SAÚL: **100 303 517**

PRIETO CEPEDA, JAVIER: **100 307 011**

ÍNDICE

TABLAS DE PRUEBAS

ÍNDICE DE GRÁFICAS

1 Introducción

En este documento se procede a la explicación de la práctica 2, *"E/S asíncrona en C"* de la asignatura Desarrollo de Software de Sistemas del grado en Ingeniería Informática de la Universidad Carlos III de Madrid. La realización de la misma la han llevado a cabo los alumnos Javier Prieto Cepeda y Saúl Alonso Monsalve.

En primer lugar, se realizará una descripción del código, tanto en la versión con ***select*** como en la versión con ***epoll***, indicando las decisiones de diseño pertinentes para la correcta implementación de la práctica.

Una vez descrito el código, se realizará una batería de pruebas para comprobar el correcto funcionamiento de la práctica. Tras esto, se procederá a comentar un análisis de los resultados obtenidos y del rendimiento conseguido, explicando los motivos de los mismos y comparando las diferencias entre las dos versiones implementadas.

A continuación, se incluye una sección para indicar cómo compilar y ejecutar el programa entregado.

Por último, se realizarán una serie de valoraciones y conclusiones sobre la realización de la práctica.

2 Descripción del código

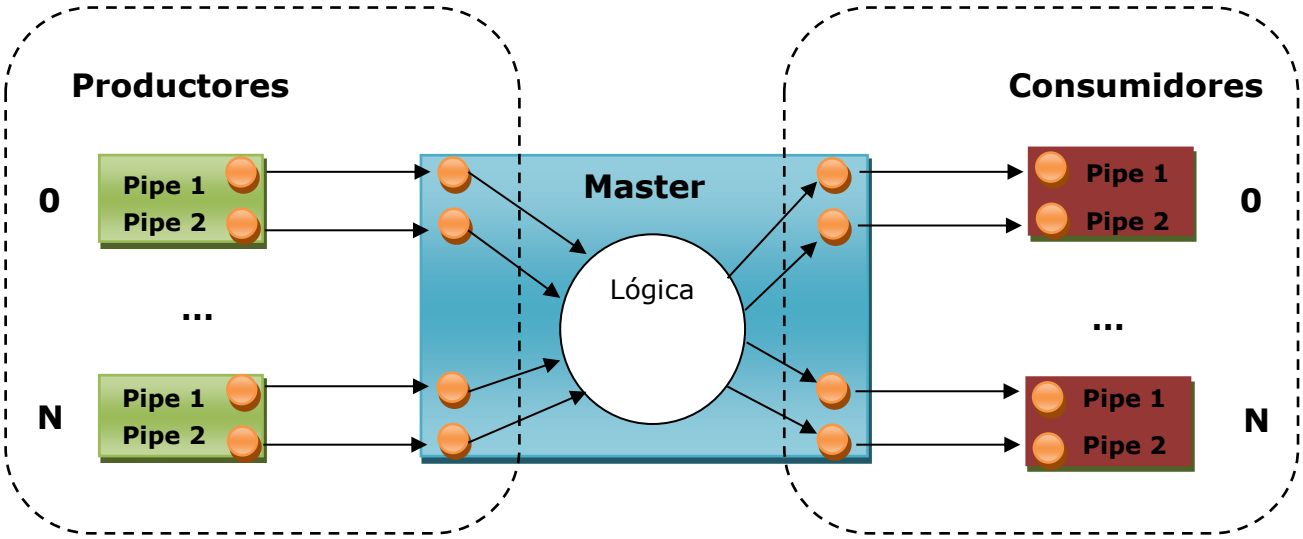
En este apartado, se va a proceder a la explicación del código implementado, tanto en la versión con ***select*** como en la versión con ***epoll***. Ambas versiones, poseen la misma estructura y finalidad, puesto que únicamente difieren en el tratamiento de los descriptores para la comunicación entre *threads*.

La implementación, consiste en un sistema productor/consumidor. Existe un conjunto de hilos productores, en donde cada hilo, tendrá comunicación directa con un número determinado de *pipes* (todos los hilos se comunicarán con el mismo número). Cada hilo, generará una serie de números aleatorios, mediante la función *rand()*, y los insertará de forma aleatoria en aquellos descriptores disponibles para la escritura del número.

Estas escrituras de número aleatorios, serán gestionadas por un único hilo (master). Este hilo, consulta qué descriptores de los pipes compartidos con el conjunto de hilos productores se encuentran listos para ser leídos, de forma que lee la información de los descriptores disponibles. A su vez, comparte una serie de pipes con un conjunto de hilos consumidores, de forma que cada vez que lee un número, consulta que descriptores de los pipes compartidos con el conjunto de hilos consumidores se encuentran disponibles, para escribir el número leído previamente.

Como se ha dicho anteriormente, existe un conjunto de hilos consumidores, en donde cada hilo comparte un número determinado de *pipes* con el hilo maestro, (todos los hilos se comunican con el mismo número de tuberías). Cada hilo, se encargará de comprobar qué descriptores de sus pipes se encuentran disponibles para ser leídos, de forma que cuando un número es escrito por el hilo master, el hilo consumidor de se encarga de leerlo.

Por tanto, el sistema consta de dos subsistemas, donde el conjunto de hilos productores y el hilo master que lee los números generados por estos forman el primer subsistema, y el hilo master que escribe los número recibidos y el conjuntos de hilos consumidores que lee los números escritos por el hilo master, forman el segundo subsistema. Este esquema de productores consumidores queda reflejado en la siguiente figura:



Como podemos observar, para un correcto funcionamiento, sería necesaria la implementación de un modelo no bloqueante, para lo cual se debe evitar el uso de llamadas *read* y *write* para la comunicación de los números entre *threads* en el sistema.

Además, se necesita de la implementación de un algoritmo equitativo, es decir, que realice el reparto de trabajo entre todos los descriptores evitando problemas de inanición.

Por tanto, se necesita de la implementación de tres funciones: slaveWriter(), master() y slaveReader(), las cuales representan cada uno de los componentes del sistema. En la implementación de estas funciones, recaen las principales diferencias en la implementación de ambas versiones, por tanto, vamos a proceder a la explicación de cada función de forma individual en cada versión.

2.1 Versión con select

La función ***slaveWriter***, será ejecutada por cada uno de los threads productores. En ella, en primer lugar, se crean las variables y estructuras necesarias para poder realizar la monitorización de los descriptores de escritura mediante la llamada select(). Para ello, es necesario obtener el máximo identificador del conjunto de descriptores que manejará cada uno de los hilos, siendo estos los descriptores de escritura de las *pipes* propias de cada hilo productor y, además, cada descriptor será añadido a una estructura de tipo fd_set, que se utilizará para la monitorización de los descriptores.

Una vez obtenidos los descriptores a monitorizar por cada hilo productor, se realiza la llamada a la función select(), en donde se pasa el máximo identificador del conjunto de descriptores más uno (variable **nfds** de la llamada *select()*), el conjunto de descriptores a monitorizar (en este caso, de escritura), y como dato a destacar, no se indica un tiempo concreto de espera para la obtención de descriptores disponibles para la realización de las escrituras de números aleatorios (queremos bloquearnos hasta que haya un descriptor disponible para escribir). Esta llamada, devuelve el número de descriptores disponibles para la realización de la escritura, dejando en la estructura de descriptores únicamente aquellos disponibles para la escritura.

La inserción de los números aleatorios en los descriptores disponibles, se realizará mediante la implementación de un buffer circular, pretendiendo evitar el problema de inanición y la aleatoriedad en las escrituras. Con este algoritmo de escritura (cada vez se intenta escribir en un descriptor distinto al de la última escritura, si es posible), conseguimos que las escrituras sean equitativas entre los diferentes pipes.

La función ***master*** será ejecutada por el hilo principal del programa. En ella, en primer lugar, se crean las variables y estructuras necesarias para poder realizar la monitorización de los descriptores (tanto de lectura como de escritura) mediante la llamada select(). Para ello, es necesario obtener el máximo identificador del conjunto de descriptores para realizar las lecturas de las pipes compartidas con el conjunto de hilos productores y, además, cada descriptor será

añadido a una estructura de tipo fd_set, que se utilizará para la monitorización de los descriptores.

Una vez obtenidos los descriptores a monitorizar (en este caso, de los pipes compartidos con los hilos productores), se realiza la llamada a la función select(), en donde se pasa el máximo identificador del conjunto de descriptores más uno, el conjunto de descriptores a monitorizar, y como dato a destacar, no se indica un tiempo concreto de espera para la obtención de descriptores disponibles para la realización de las lecturas de los números (queremos bloquearnos hasta que haya un descriptor disponible para leer). Esta llamada, devuelve el número de descriptores disponibles para la realización de la lectura, dejando en la estructura de descriptores únicamente aquellos disponibles para la lectura.

Una vez obtenidos los descriptores disponibles para la lectura, se itera por cada uno de ellos, de forma que cada vez que se obtiene un descriptor disponible, se realiza la lectura del número escrito por el conjunto de hilos productores. Tras la lectura del número, se crean las variables y estructuras necesarias para poder realizar la monitorización de los descriptores para realizar la escritura en las pipes compartidas con el conjunto de hilos consumidores mediante la llamada select(). Antes de eso, se elige el hilo en el que escribir realizando la operación **elemento_leido%numero_hilos_lectores**. Tras esto, es necesario obtener el máximo identificador del conjunto de descriptores para realizar las escrituras en las pipes compartidas con el conjunto de hilos consumidores, y además, cada descriptor será añadido a una estructura de tipo fd_set, que se utilizará para la monitorización de los descriptores.

Una vez obtenidos los descriptores a monitorizar, se realiza la llamada a la función select(), en donde se pasa el máximo identificador del conjunto de descriptores más uno, el conjunto de descriptores a monitorizar, y como dato a destacar, no se indica un tiempo concreto de espera para la obtención de descriptores disponibles para la realización de las escrituras de los números (queremos bloquearnos hasta que haya un descriptor disponible para escribir). Esta llamada, devuelve el número de descriptores disponibles para la realización de la escritura, dejando en la estructura de descriptores únicamente aquellos disponibles para la escritura.

Una vez obtenidos los descriptores disponibles para la escritura del número, se obtiene mediante la función *rand()* de forma aleatoria el pipe en el cual se escribirá el número, realizándose la escritura de éste y tratando así de no escribir siempre en el mismo pipe.

La función ***slaveReader***, será ejecutada por cada uno de los threads consumidores. En ella, en primer lugar, se crean las variables y estructuras necesarias para poder realizar la monitorización de los descriptores mediante la llamada select(). Para ello, es necesario obtener el máximo identificador del conjunto de descriptores que manejará cada uno de los hilos, siendo estos los descriptores de lectura de las *pipes* propias de cada hilo consumidor y además, cada descriptor será añadido a una estructura de tipo fd_set, que se utilizará para la monitorización de los descriptores.

Una vez obtenidos los descriptores a monitorizar por cada hilo consumidor, se realiza la llamada a la función select(), en donde se pasa el máximo identificador del conjunto de descriptores más uno, el conjunto de descriptores a monitorizar, y como dato a destacar, no se indica un tiempo concreto de espera para la obtención de descriptores disponibles para la realización de las lecturas de los números (queremos bloquearnos hasta que haya un descriptor disponible para leer). Esta llamada, devuelve el número de descriptores disponibles para la realización de la lectura, dejando en la estructura de descriptores únicamente aquellos disponibles para la lectura.

La lectura de los números de los descriptores disponibles, se realizará mediante la implementación de un buffer circular (cada vez se intenta leer de un descriptor distinto al de la última lectura, si es posible), con lo que conseguimos que las lecturas sean equitativas entre los diferentes pipes.

2.2 Versión con epoll

La función ***slaveWriter***, será ejecutada por cada uno de los threads productores. En ella, en primer lugar se crea la instancia a epoll. Una vez creada la instanciación de epoll, se añaden a la estructura de epoll (del tipo epoll_event) los descriptores a monitorizar, indicando el identificador del descriptor, el tipo de monitorización a realizar (en este caso EPOLLOUT, de escritura). Se añade cada descriptor a la estructura mediante la función epoll_ctl, con el flag EPOLL_CTL_ADD, indicando que añada el descriptor y la información indicada anteriormente a la estructura de monitorización de epoll.

Una vez añadidos todos los descriptores a monitorizar, se realizará la generación de los números aleatorios. En primer lugar, se realizará la llamada a la función epoll_wait() para actualizar los descriptores disponibles para la realización de las escrituras. Esta función, retornará el número de descriptores disponibles para escribir. Al igual que en la versión anterior, no indicamos un tiempo concreto de espera para la obtención de descriptores disponibles para escritura (indicamos -1, ya que queremos bloquearnos hasta que haya un descriptor disponible para escribir).

Una vez obtenidos los descriptores listos como consecuencia de las escrituras en el hilo productor, procedemos a la generación de números aleatorios. El comienzo de la lista, se realizará de manera aleatoria, simulando un buffer circular, de modo que se evite una posible inanición de descriptores e intentando repartir de forma equitativa el trabajo en cada pipe. Se irá recorriendo el buffer, escribiendo los números aleatorios, generados con la función *rand().*

La función ***master*** será ejecutada por el hilo principal del programa. En ella, en primer lugar se crean las instancias de epoll tanto para la realización de las lecturas de las pipes compartidos con el conjunto de threads productores como para la realización de las escrituras en las pipes compartidas con el conjunto de threads consumidores. Una vez creadas las instacias epoll, se montan las estructuras necesarias para la monitorización de los descriptores.

Es importante tener en cuenta, que la función master realiza dos monitorizaciones (lectura del conjunto productor y escritura en el conjunto consumidor). En primer lugar, se monta la estructura necesaria para la realización de las lecturas, en donde se añaden los

descriptores a monitorizar y el tipo de monitorización deseado de cada uno, en este caso, de lectura (EPOLLIN). Una vez montada toda la estructura para la monitorización de las lecturas, procedemos a montar la estructura para la monitorización de las escrituras. Se realizará de la misma manera que para las lecturas, salvo que en este caso, los descriptores serán los descriptores de escritura de los pipes compartidas con el conjunto de threads consumidores, y el atributo de monitorización EPOLLOUT, que indica la disponibilidad para escritura.

Una vez montadas las estructuras de monitorización, se realiza en primer lugar la llamada a la función epoll_wait() para la obtención de los descriptores disponibles de lectura. Una vez obtenidos los descriptores disponibles, se itera mediante la simulación de un buffer circular por los descriptores de lectura disponibles de forma que cada vez que se obtiene un dato, éste debe de ser escrito en una de los pipes compartidas con el conjunto de threads consumidores. De esta manera, una vez leído un dato, se realiza una llamada a la función epoll_wait() para obtener los descriptores disponibles para la realización de las escrituras. Una vez obtenidos los descriptores disponibles para la realización de las escrituras, se escogen los descriptores correspondientes a los pipes que el master comparte con el hilo de lectura **elemento_leido%numero_hilos_lectores** y se itera por cada uno de estos descriptores, de forma que iteramos por los distintos descriptores hasta que insertamos el número en uno disponible para la escritura. Una vez escrito, se vuelve a comenzar con el proceso de leer otro número y escribirlo nuevamente en otra tubería.

La función ***slaveReader***, será ejecutada por cada uno de los threads consumidores. En ella, en primer lugar se crea la instancia a epoll, pasando como parámetro el número de pipes que tendrá el hilo. Una vez creada la instanciación de epoll, se añaden a la estructura de epoll los descriptores a monitorizar, indicando el identificador del descriptor, el tipo de monitorización a realizar (en este caso EPOLLIN, de lectura). Se añade cada descriptor a la estructura mediante la función epoll_ctl, con el flag EPOLL_CTL_ADD, indicando que añada el descriptor y la información indicada anteriormente a la estructura de monitorización de epoll.

Una vez añadidos todos los descriptores a monitorizar, se realizará la lectura de los pipes disponibles para ser leídas. En primer lugar, se

realizará la llamada a la función epoll_wait() para actualizar los descriptores disponibles para la realización de las lecturas. Esta función, retornará el número de descriptores disponibles para leer.

Una vez obtenidos los descriptores disponibles para la realización de las lecturas, procedemos a leer los números. La lectura se realizará simulando el funcionamiento de un buffer circular, obteniendo la cabeza del buffer de manera aleatoria, para evitar una posible inanición en las lecturas, y repartir de forma equitativa las lecturas de las pipes del hilo.

2.3 Decisiones de diseño

En cuanto a las decisiones de diseño tomadas, es necesario destacar los siguientes aspectos, justificando los motivos por los cuales se han llevado a cabo las siguientes implementaciones:

- Se ha decidido realizar un algoritmo equitativo a la hora de recorrer los descriptores en todos los subsistemas (hilos productores, maestro y consumidores), para evitar una posible inanición a la hora de dar trabajo a las tuberías y a la hora de obtener los números en las lecturas, de forma que todos los descriptores sean usados prácticamente por igual. Para lograrlo, se ha implementado la simulación de un buffer circular, en donde el comienzo de la iteración se asigna de manera aleatoria, tanto en la versión con select() como en la versión con epoll().
- Se ha decidido implementar una monitorización de los descriptores en los que escribir en los hilos productores para nunca intentar escribir en una tubería llena y además tener más facilidades para implementar nuestro algoritmo equitativo.
- Se ha decidido implementar una monitorización de los descriptores de los que leer en los hilos consumidores para nunca intentar leer de una tubería llena y además tener más facilidades para implementar nuestro algoritmo equitativo.

3 Batería de pruebas

En este apartado vamos a presentar nuestra batería de pruebas para tratar de demostrar el correcto funcionamiento del sistema. El objetivo de esta sección es corroborar que se cumple el funcionamiento básico de las dos versiones (escrituras, lecturas y monitorización del maestro correctos). Las pruebas de rendimiento se corresponden con la sección 4: Evaluación del rendimiento: select VS epoll.

PRUEBA 1	
Objetivo	Correcto funcionamiento de la versión select. Demostrar que el flujo de datos siempre ocurre del siguiente orden: - Hilo escritor escribe elemento X en tubería 1. - Maestro lee elemento X de tubería 1. - Maestro escribe elemento X en tubería 2. - Hilo lector lee elemento X de tubería 2. Además, se debe comprobar que la tubería 1 corresponde a los pipes que el maestro comparte con los hilos de escritura y la tubería 2 la tubería que el maestro comparte con los hilos de escritura.
Entradas	Modelo inicial: - Número de hilos escritores: 2 - Número de hilos lectores: 2 - Número de tuberías por hilo: 2
Salida	Como se puede ver en rojo en la captura de pantalla, ocurre lo siguiente: - Hilo escritor escribe elemento 1830717880 en tubería 1 (descriptor 4). - Maestro lee elemento 1830717880 de tubería 1 (descriptor 3). - Maestro escribe elemento 1830717880 en tubería 6 (descriptor 14). - Hilo lector lee elemento 1830717880 de tubería 6 (descriptor 13).

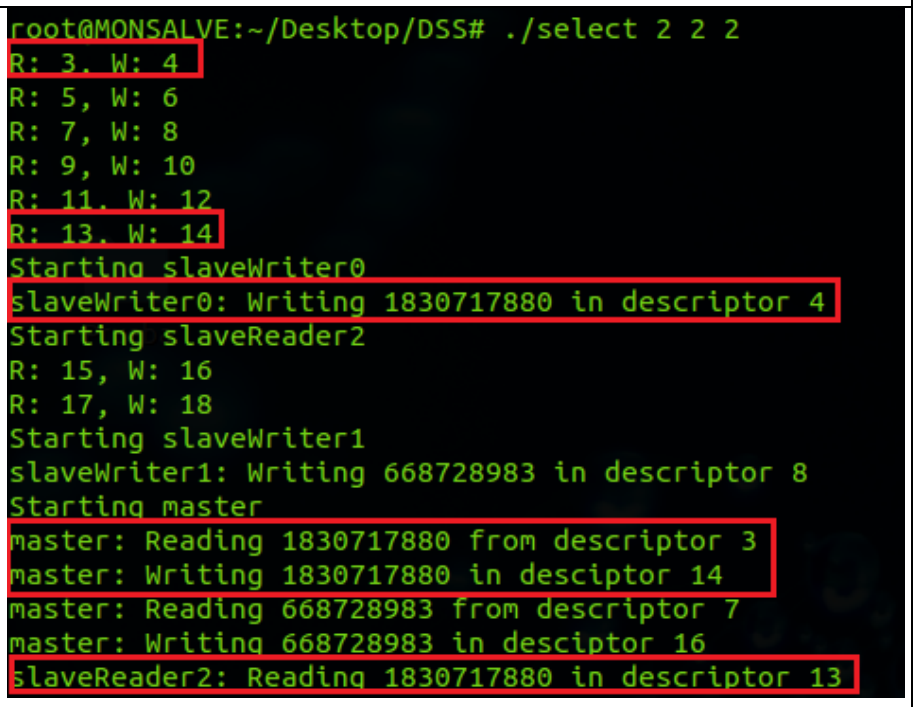
PRUEBA 1	
Conclusión	Como se ha demostrado, se cumple el resultado esperado.
Captura de pantalla	

Tabla 1: Prueba 1

PRUEBA 2	
Objetivo	Correcto funcionamiento de la versión select. Demostrar que el flujo de datos siempre ocurre del siguiente orden: - Hilo escritor escribe elemento X en tubería 1. - Maestro lee elemento X de tubería 1. - Maestro escribe elemento X en tubería 2. - Hilo lector lee elemento X de tubería 2. Además, se debe comprobar que la tubería 1 corresponde a los pipes que el maestro comparte con los hilos de escritura y la tubería 2 la tubería que el maestro comparte con los hilos de escritura.
Entradas	Modelo inicial: - Número de hilos escritores: 1 - Número de hilos lectores: 1 - Número de tuberías por hilo: 6
Salida	Como se puede ver en rojo en la captura de pantalla, ocurre lo siguiente: - Hilo escritor escribe elemento 596093787 en tubería 2 (descriptor 6). - Maestro lee elemento 596093787 de tubería 2 (descriptor 5). - Maestro escribe elemento 596093787 en tubería 9 (descriptor 20). - Hilo lector lee elemento 596093787 de tubería 9 (descriptor 19).
Conclusión	Como se ha demostrado, se cumple el resultado esperado.

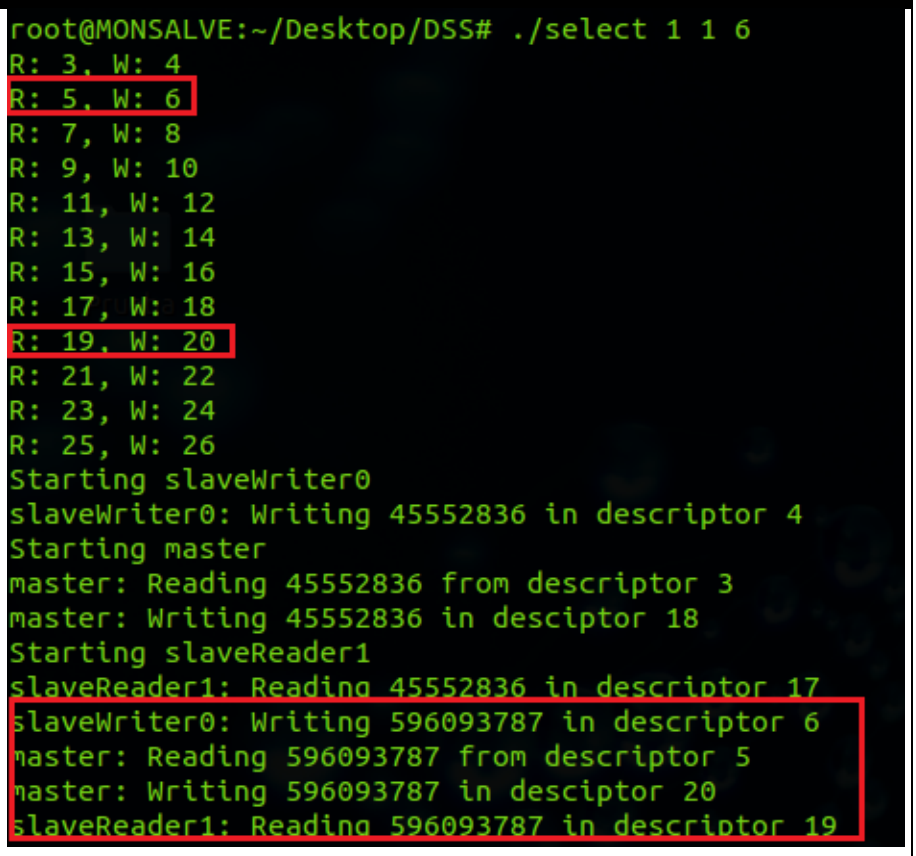
PRUEBA 2	
Captura de pantalla	

Tabla 2: Prueba 2

PRUEBA 3	
Objetivo	Correcto funcionamiento de la versión epoll. Demostrar que el flujo de datos siempre ocurre del siguiente orden: - Hilo escritor escribe elemento X en tubería 1. - Maestro lee elemento X de tubería 1. - Maestro escribe elemento X en tubería 2. - Hilo lector lee elemento X de tubería 2. Además, se debe comprobar que la tubería 1 corresponde a los pipes que el maestro comparte con los hilos de escritura y la tubería 2 la tubería que el maestro comparte con los hilos de escritura.
Entradas	Modelo inicial: - Número de hilos escritores: 2 - Número de hilos lectores: 2 - Número de tuberías por hilo: 2
Salida	Como se puede ver en rojo en la captura de pantalla, ocurre lo siguiente: - Hilo escritor escribe elemento 49341916 en tubería 3 (descriptor 8). - Maestro lee elemento 49341916 de tubería 3 (descriptor 7). - Maestro escribe elemento 49341916 en tubería 6 (descriptor 14). - Hilo lector lee elemento 49341916 de tubería 6 (descriptor 13).
Conclusión	Como se ha demostrado, se cumple el resultado esperado.

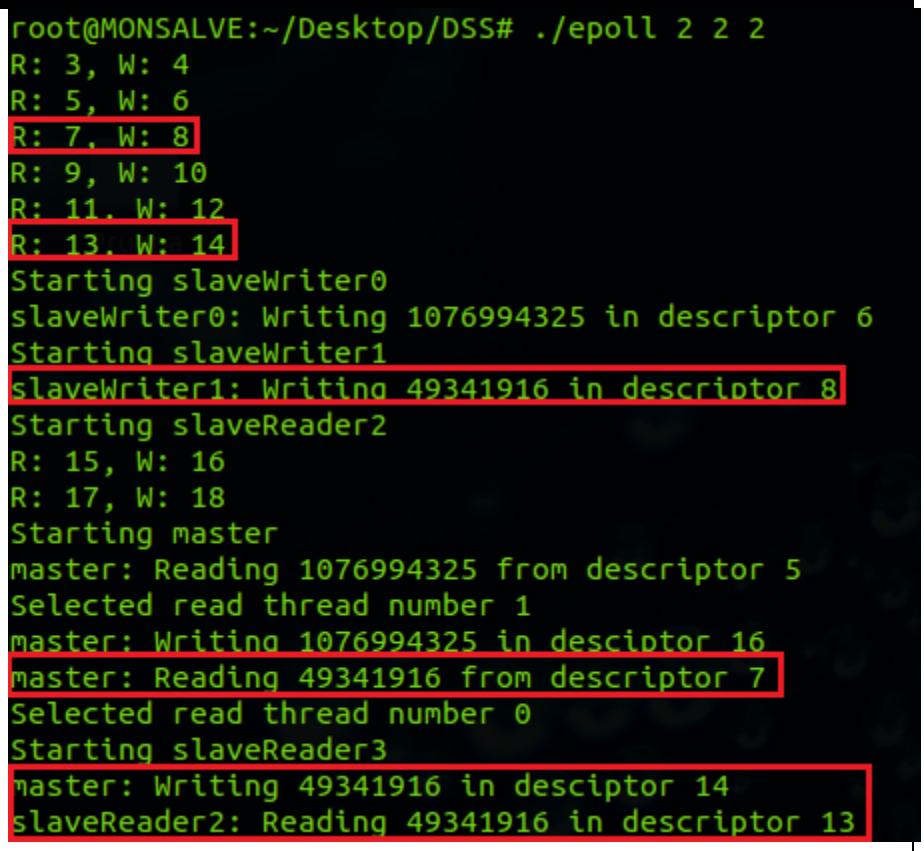
PRUEBA 3	
Captura de pantalla	

Tabla 3: Prueba 3

PRUEBA 4	
Objetivo	Correcto funcionamiento de la versión epoll. Demostrar que el flujo de datos siempre ocurre del siguiente orden: - Hilo escritor escribe elemento X en tubería 1. - Maestro lee elemento X de tubería 1. - Maestro escribe elemento X en tubería 2. - Hilo lector lee elemento X de tubería 2. Además, se debe comprobar que la tubería 1 corresponde a los pipes que el maestro comparte con los hilos de escritura y la tubería 2 la tubería que el maestro comparte con los hilos de escritura.
Entradas	Modelo inicial: - Número de hilos escritores: 1 - Número de hilos lectores: 1 - Número de tuberías por hilo: 6
Salida	Como se puede ver en rojo en la captura de pantalla, ocurre lo siguiente: - Hilo escritor escribe elemento 1273661342 en tubería 4 (descriptor 10). - Maestro lee elemento 1273661342 de tubería 4 (descriptor 9). - Maestro escribe elemento 1273661342 en tubería 7 (descriptor 16). - Hilo lector lee elemento 1273661342 de tubería 7 (descriptor 15).
Conclusión	Como se ha demostrado, se cumple el resultado esperado.

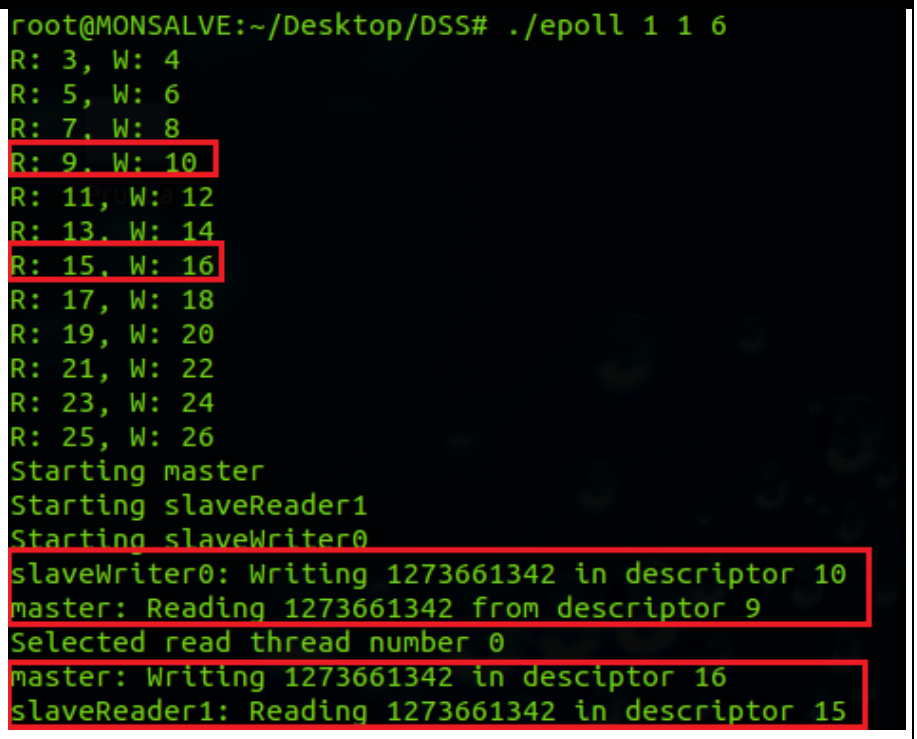
PRUEBA 4	
Captura de pantalla	

Tabla 4: Prueba 4

4 Evaluación del rendimiento: select VS epoll

En este apartado se pretende comparar las dos alternativas implementadas, ***select*** y ***epoll***, en términos de rendimiento. Para ello, hemos realizado tres casos de estudio diferentes. En cada uno de ellos, primero se expone una breve descripción del caso de estudio, seguido de los resultados adquiridos en forma de tablas y gráficas, junto con unas conclusiones detalladas que tratan de comentar tanto los resultados obtenidos como sus explicaciones teóricas.

Nota: para cada prueba, los tiempos presentados en las tablas recogidas en este apartado se han obtenido tras realizar la media aritmética de diez ejecuciones diferentes. Además, para comparar los tiempos de una forma más exacta, hemos hecho que los esclavos no duerman tras escribir/leer en las tuberías.

4.1 Variación del número de descriptores totales

En esta prueba vamos a tratar de comparar el rendimiento de las versiones con ***select*** y con ***epoll*** realizando pruebas en las que solo existe un hilo esclavo de escritura y otro de lectura, pero el número de tuberías por hilo aumenta gradualmente (potencias de dos).

Parámetros comunes en todas las pruebas:

- **Número de escritores (W)**: 1
- **Número de lectores (R)**: 1

Parámetros que varían en las pruebas:

- **Número de tuberías por hilo (PPT):** 2^0, 2^1, 2^2,…, 2^{12}.

Resultados:

- Tiempo en milisegundos en leerse 10000 elementos de en los hilos de lectura para las versiones ***select*** y ***epoll***. Es decir, el tiempo que transcurre desde el comienzo de la ejecución hasta que los hilos de lectura leen un total de 10000 elementos que el hilo maestro ha escrito en las tuberías que comparte con estos hilos, habiendo sido estos elementos previamente leídos por el maestro de las tuberías que comparte con los hilos de escritura.

Tabla con los datos:

Escritores, Lectores, Tuberias por hilo	Select (ms)	Epoll (ms)
W = 1, R = 1, PPT = 1	21	22
W = 1, R = 1, PPT = 2	31	23
W = 1, R = 1, PPT = 4	32	22
W = 1, R = 1, PPT = 8	31	26
W = 1, R = 1, PPT = 16	33	25
W = 1, R = 1, PPT = 32	33	24
W = 1, R = 1, PPT = 64	65	27
W = 1, R = 1, PPT = 128	111	44
W = 1, R = 1, PPT = 256	206	70
W = 1, R = 1, PPT = 512	381	133
W = 1, R = 1, PPT = 1024	833	222
W = 1, R = 1, PPT = 2048	3052	603
W = 1, R = 1, PPT = 4096	16485	1125

Tabla 5: Tiempos 1

Gráfica comparativa de tiempos:

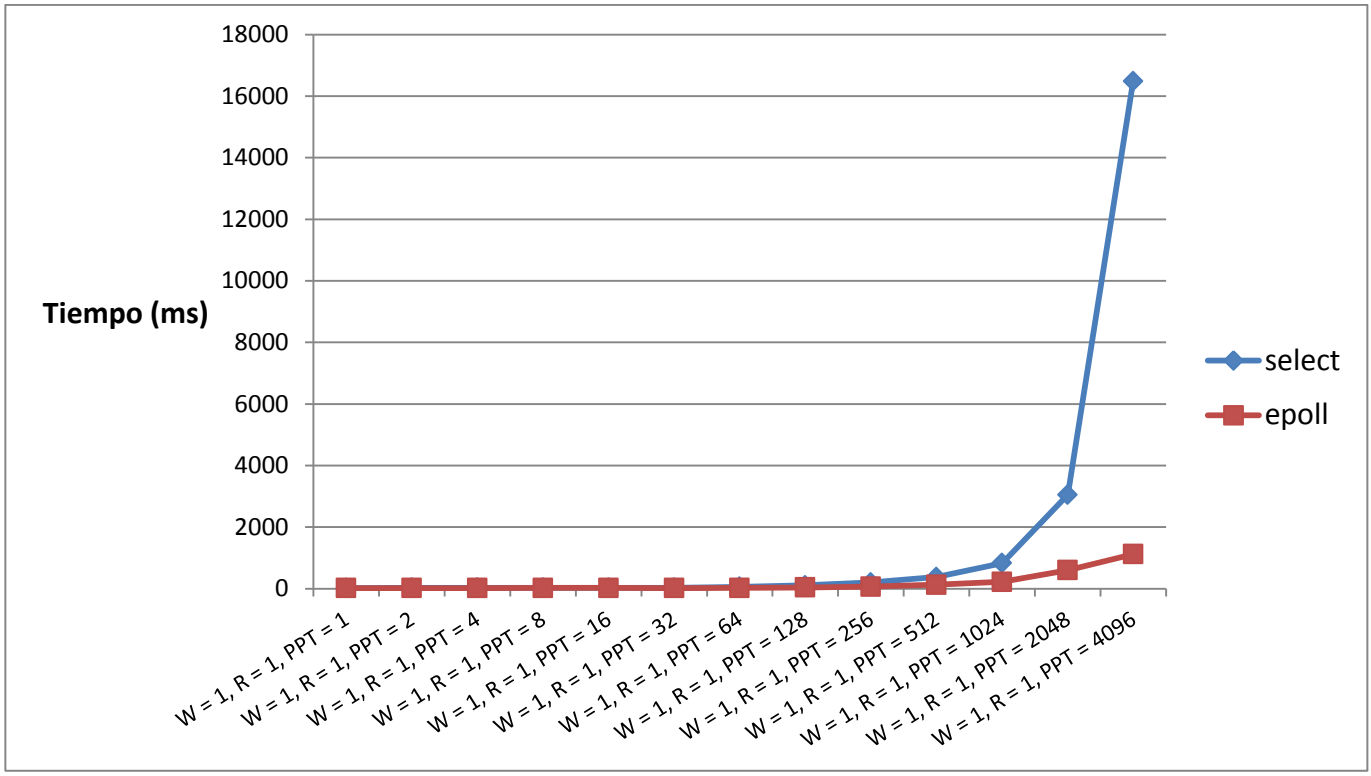


Ilustración 1: Gráfica rendimiento 1

En esta prueba se ha ido aumentando progresivamente el número de tuberías por hilo, pero el número de esclavos tanto de lectura como de escritura se ha mantenido estable. Esto significa que el número de lecturas y escrituras se ha mantenido constante (el número de hilos no cambia), pero sí se ha aumentado con cada prueba el número de descriptores a monitorizar.

Como se puede apreciar en la gráfica anterior, al aumentar el número de descriptores totales a monitorizar (número de tuberías por hilo), la versión con ***select*** aumenta de una manera mucho más severa que la versión con ***epoll***, ya que la complejidad de ***select*** es proporcional al número de descriptores. Sin embargo, en la versión con ***epoll*** el kernel mantiene el estado de la llamada de ***epoll*** y la complejidad de éste es proporcional al número de señales recibidas. Esta es la razón por la que el tiempo en la versión de ***epoll*** se mantiene más estable al aumentar el número de descriptores.

4.2 Variación del número de esclavos

En esta prueba vamos a tratar de comparar el rendimiento de las versiones con ***select*** y con ***epoll*** realizando pruebas en las que el número de descriptores de fichero (número de tuberías) se mantiene constante, pero se aumenta el número de hilos lectores y escritores gradualmente (potencias de dos).

Parámetros comunes en todas las pruebas:

- **Número de tuberías totales del sistema:** Siempre 8192 tuberías, que se obtiene al multiplicar realizar la operación:

$$(\text{Número de hilos escritores} + \text{Número de hilos lectores}) * \text{Número de tuberías por hilo}$$

Parámetros que varían en las pruebas:

- **Número de hilos de escritura (W):** 2^0, 2^1, 2^2, 2^3, 2^4, 2^5.
- **Número de hilos de lectura (R):** 2^0, 2^1, 2^2, 2^3, 2^4, 2^5.
- **Número de tuberías por hilo (PPT):** 8192, 2048, 1024, 512, 256, 128.

Resultados:

- Tiempo en milisegundos en leerse 10000 elementos de en los hilos de lectura para las versiones ***select*** y ***epoll***. Es decir, el tiempo que transcurre desde el comienzo de la ejecución hasta que los hilos de lectura leen un total de 10000 elementos que el hilo maestro ha escrito en las tuberías que comparte con estos hilos, habiendo sido estos elementos previamente leídos por el maestro de las tuberías que comparte con los hilos de escritura.

Tabla con los datos:

Escritores, Lectores, Tuberias por hilo	Select (ms)	Epoll (ms)
W = 1, R = 1, PPT = 8192	46785	11427
W = 2, R = 2, PPT = 2048	8614	4550
W = 4, R = 4, PPT = 1024	3100	1248
W = 8, R = 8, PPT = 512	2675	447
W = 16, R = 16, PPT = 256	1602	1315
W = 32, R = 32, PPT = 128	468	2913

Tabla 6: Tiempos 2

Gráfica comparativa de tiempos:

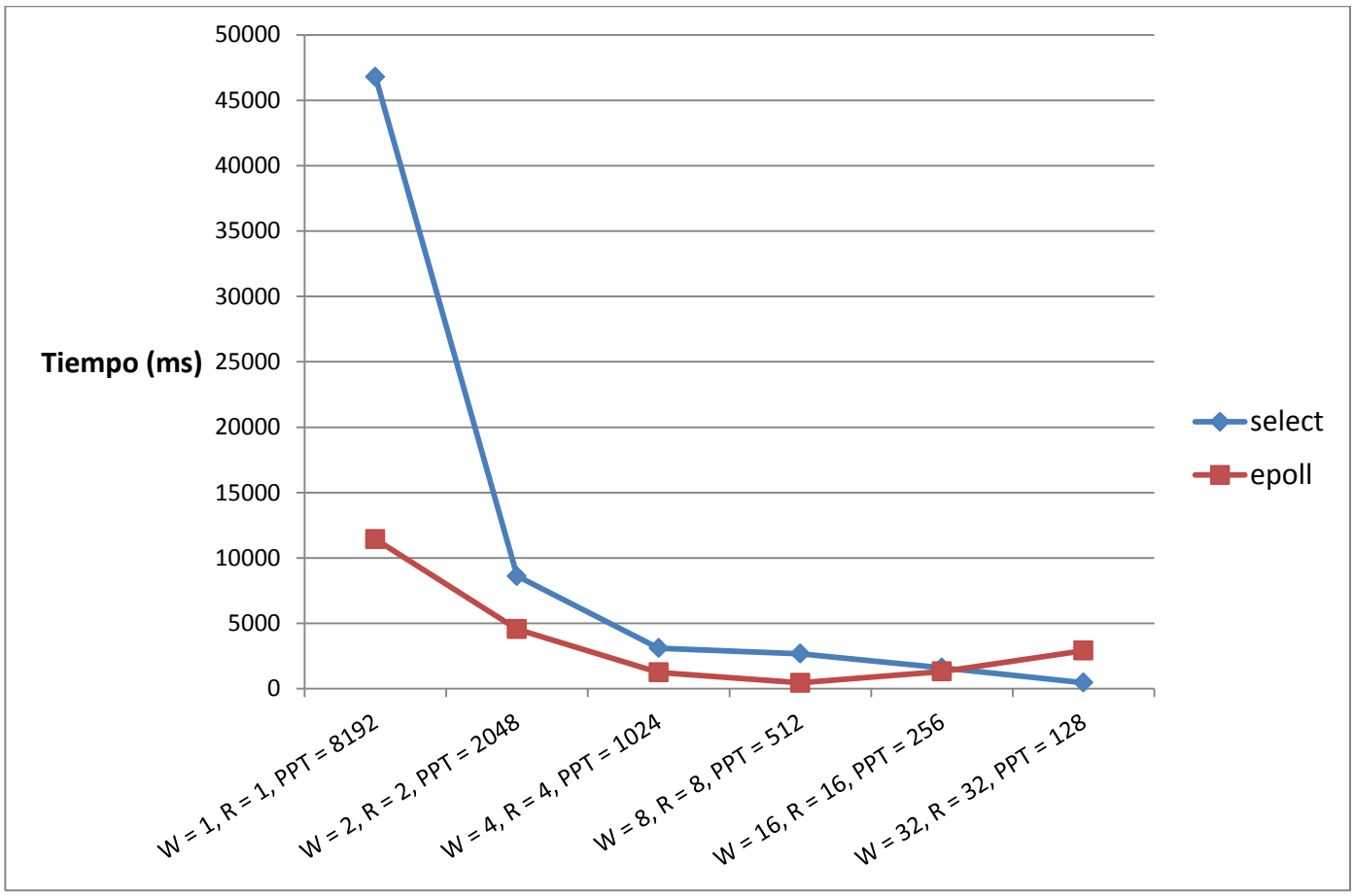


Ilustración 2: Gráfica rendimiento 2

En esta prueba se ha ido aumentando progresivamente el número de hilos de lectura y escritura, a la vez que reduciendo el número de tuberías por hilo, siempre procurando mantener constante el número total de tuberías (en 8192). Como el número de tuberías se mantiene estable y es idéntico entre lectores y escritores, el número de descriptores de fichero también es constante.

Como se puede apreciar en la gráfica anterior, al aumentar el número de lectores y escritores, el tráfico de elementos es mucho más fluido para la versión con ***select***. Esto se debe a que, al no aumentar el número de descriptores totales a monitorizar, el tiempo solo depende de la frecuencia de lecturas y escrituras. Obviamente, cuantos más hilos, más escrituras y lecturas y, por lo tanto, antes se llega al final

de cada prueba, que consiste en que los hilos lectores lean 10000 elementos, como hemos explicado previamente.

Con la versión de ***epoll*** pasa algo parecido. Cuantos más hilos de lectura y escritura, mejor es el tráfico y el tiempo baja. Sin embargo, cabe destacar que la complejidad de ***epoll*** depende del número de señales, ya que monitoriza el conjunto de descriptores recibiendo señales por cada evento y actuando en consecuencia. En esta prueba, al aumentar el número de lectores y escritores demasiado, como se puede ver en la gráfica anterior, se produce una subida de tiempos final en el ejemplo con ***epoll***. Esto se debe a que se reciben excesivas señales, lo cual empeora el rendimiento de ***epoll*** y además contrarresta el efecto positivo del gran tráfico de elementos con un número de esclavos alto.

4.3 Variación del número de esclavos y de tuberías totales

En esta prueba vamos a tratar de comparar el rendimiento de las versiones con ***select*** y con ***epoll*** realizando pruebas en las que se aumenta tanto el número de esclavos totales (de escritura y de lectura) como fichero (número de tuberías) del sistema. Con esta prueba queremos aunar las partes variantes de las dos pruebas anteriores y comparar qué ocurre.

Parámetros comunes en todas las pruebas:

- **Número de tuberías por hilo (PPT):** 512.

Parámetros que varían en las pruebas:

- **Número de hilos de escritura (W):** 2^0, 2^1, 2^2, 2^3, 2^4.
- **Número de hilos de lectura (R):** 2^0, 2^1, 2^2, 2^3, 2^4.

Resultados:

- Tiempo en milisegundos en leerse 10000 elementos de en los hilos de lectura para las versiones ***select*** y ***epoll***. Es decir, el tiempo que transcurre desde el comienzo de la ejecución hasta que los hilos de lectura leen un total de 10000 elementos que el hilo maestro ha escrito en las tuberías que comparte con estos hilos, habiendo sido estos elementos previamente leídos por el maestro de las tuberías que comparte con los hilos de escritura.

Tabla con los datos:

Escritores, Lectores, Tuberias por hilo	Select (ms)	Epoll (ms)
W = 1, R = 1, PPT = 512	297	132
W = 2, R = 2, PPT = 512	1207	230
W = 4, R = 4, PPT = 512	2795	502
W = 8, R = 8, PPT = 512	18426	9283
W = 16, R = 16, PPT = 512	99145	39001

Tabla 7: Tiempos 3

Gráfica comparativa de tiempos:

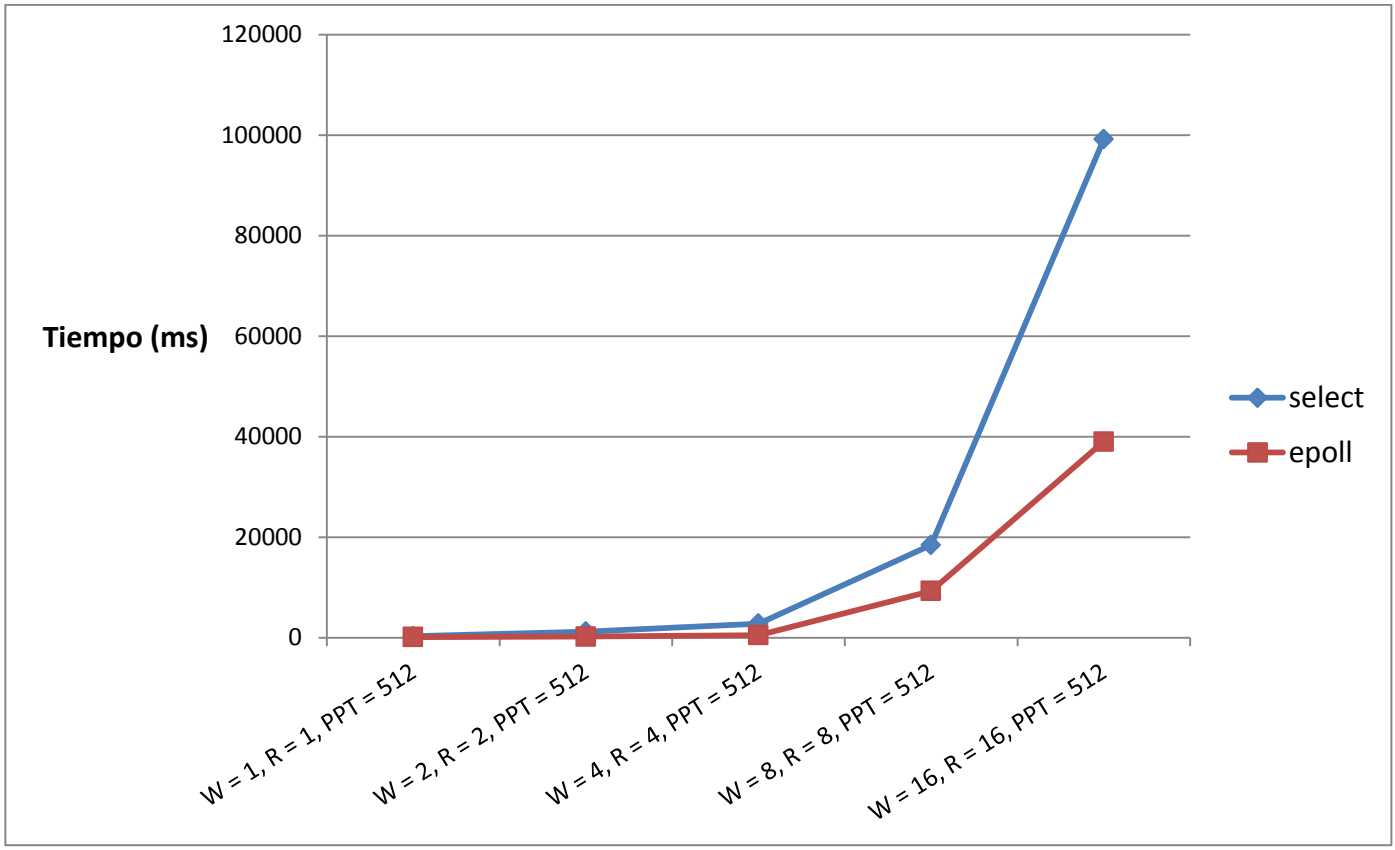


Ilustración 3: Gráfica rendimiento 3

En esta prueba se ha ido aumentando progresivamente el número de hilos de lecturas y escritura, manteniendo el número de tuberías constante. Con esto conseguimos un tráfico de elementos más fluido cada vez que aumentamos el número de esclavos, a la vez que aumentamos el número total de tuberías (si mantenemos constante el número de tuberías por hilo y aumentamos los hilos, el número de tuberías totales aumenta).

Las conclusiones que se pueden extraer de estas pruebas son claras. En la versión con ***select***, cada prueba es más costosa en cuanto a tiempo ya que, como se ha explicado antes, la complejidad de ***select*** aumenta con el número de descriptores a monitorizar. En cada prueba multiplicamos el número de tuberías totales y, por lo tanto, también el número de descriptores de fichero a monitorizar.

En el caso de la versión con ***epoll***, al aumentar cada vez más el número de esclavos (escritores y lectores) hay mucho más tráfico de datos y, por lo tanto, también más señales. Como se ha explicado

previamente, la complejidad de ***epoll*** depende en el número de señales y no en el número de descriptores, lo que constata nuestros resultados.

Con esta prueba, hemos conseguido unir en el mismo caso de estudio unos parámetros que ponen de manifiesto las virtudes y los defectos de las dos alternativas analizadas en esta práctica, ***select*** y ***epoll***.

5 Compilación y ejecución del programa

Para compilar el programa, se ha incluido en el código un fichero Makefile, por lo que basta con ejecutar el comando ***make***. Con esto, se generarán los ficheros ***select*** y ***epoll***.

Una vez hecho esto, las versiones utilizando ***select*** y ***epoll*** se ejecutarán con el comando:

./PROGRAMA N_HILOS_ESCRITURA N_HILOS_LECTURA N_TUBERIAS_POR_HILO

Donde:

- **PROGRAMA**: es el nombre del fichero ejecutable (select o epoll).
- **N_HILOS_ESCRITURA**: Número de hilos de escritura.
- **N_HILOS_LECTURA**: Número de hilos de lectura.
- **N_TUBERIAS_POR_HILO**: Número de tuberías por hilo.

NOTA: Si se desea que el programa imprima trazas, se debe descomentar la línea *//#define DEBUG* en los ficheros *select.c* y *epoll.c*.

6 Conclusiones

La práctica ha servido para conocer la diferencia entre ambos sistemas de monitorización de descriptores para la realización de entrada salida, demostrando la mejora en tiempos obtenida por el sistema de señales que implementa la llamada **epoll** sobre la llamada **select**. También resulta interesante destacar de la necesidad de no realizar las pruebas en entornos compartidos, debido a que la medición de rendimiento en estos no sería real, ya que el número de interrupciones generados por las señales puede verse alterado por procesos externos.

Otro dato a destacar, es la complejidad en tiempos que implica select, puesto que su crecimiento es proporcional al número de descriptores a monitorizar. Puede no resultar atractivo en cuanto a la diferencia de rendimiento frente a epoll, pero según en el contexto de su implementación, puede resultar interesante y válido frente a epoll.

Uno de los problemas encontrados en la implementación del sistema, ha sido la necesidad de simular buffers circulares para la iteración por la lista de descriptores. El motivo de esta simulación, se debe a evitar una posible inanición en los descriptores, además de dotar de aleatoriedad a la cabeza del buffer, con la consecuente mejora en reparto de trabajo a las pipes. Gracias a ello, obtenemos un mejor rendimiento y un reparto de trabajo más equitativo.