elektron ámbar

Original Paper

Revista elektron, Vol. 9, No. 1, pp. 23-29 (2025)
https://doi.org/10.37537/rev.elektron.9.1.212.2025

Sistema de archivos para memorias flash embebidas del tipo NAND

File system for embedded NAND flash memory

Germán Castro

Laboratorio de Sistemas Embebidos
Facultad de Ingeniería - UBA
Buenos Aires, Argentina
gcastro@fi.uba.ar

Recibido: 05/04/25; Aceptado: 02/06/25

***Resumen*—En este trabajo se presenta el desarrollo e implementación de un sistema de archivos para memorias flash embebidas del tipo NAND, desarrollado para configurar dinámicamente transceptores ópticos de alta velocidad utilizados en la comunicación entre centros de datos, está basado en una política de *Copy-on-Write* y optimizado para aportar mínima sobrecarga de código. El sistema de archivos desarrollado ocupó un espacio de memoria de 3.084 bytes en ROM y 340 bytes en RAM.**

Palabras clave: BPM; COW; FS; GC; NAND; WL.

***Abstract*—This work presents the development and implementation of a file system for embedded NAND flash memories, aimed at dynamically configuring high-speed optical transceivers used in data center communication, based on a Copy-on-Write policy and optimized to provide minimal code overhead. The developed file system occupied a memory space of 3,084 bytes in ROM and 340 bytes in RAM.**

Keywords: BPM; COW; FS; GC; NAND; WL.

I. Introducción

Las memorias flash presentan ventajas como la no-volatilidad, alta densidad, bajo consumo de energía y bajo costo, por estos motivos son las memorias utilizadas para dispositivos embebidos pequeños y portables. Sin embargo, estas memorias realizan operaciones de escritura de manera mucho más lenta que las memorias volátiles, tales como DRAM, y requieren una operación previa de borrado de la página que se desea escribir, la cual consume una gran cantidad de tiempo. Adicionalmente, las páginas que constituyen una memoria flash poseen un número finito de operaciones de borrado soportadas, por lo que estas operaciones no solo deben ser evitadas siempre que sea posible, sino que también requieren algoritmos sofisticados para distribuirlas de manera uniforme a lo largo de toda la memoria y así prolongar la vida útil de la misma.

Todos estos motivos hacen que los sistemas de archivos convencionales en sistemas operativos, tales como FAT, ext o NTFS, no resulten óptimos para ser implementados en una memoria flash embebida y se termine recurriendo, en la mayoría de los casos, a un manejo personalizado de la memoria como única solución.

La robustez del sistema de archivos empieza a tener relevancia cuando se contempla lo común que resulta en los sistemas embebidos la pérdida de alimentación de energía en cualquier momento. Usualmente el firmware de un microcontrolador es simple y reactivo, sin concepto alguno sobre una rutina de apagado. Esto presenta un gran desafío para un almacenamiento persistente, donde una desafortunada pérdida de energía puede corromper una parte o la totalidad del almacenamiento y dejar el dispositivo en un estado irrecuperable.

El propósito de este trabajo fue desarrollar un sistema de archivos (FS) para memorias flash embebidas del tipo NAND, optimizado para minimizar la sobrecarga de la aplicación en la que se integra. El sistema permitió almacenar y persistir archivos en memoria no volátil teniendo especial consideración en la reducción del desgaste de los bloques y en evitar posibles corrupciones de los datos, ayudando así a extender la vida útil de dicha memoria. El fin último es poder utilizar el sistema de archivos desarrollado para almacenar y manejar dinámicamente las configuraciones y calibraciones de los transceptores ópticos de alta velocidad utilizados para la comunicación entre centros de datos.

I-A. Sistema de archivos basado en copia al escribir

Los FSs basados en copia al escribir (COW) son muy similares a otros FSs basados en páginas, pero en lugar de actualizar la página en su lugar, todas las actualizaciones se realizan creando una copia con los cambios y reemplazando cualquier referencia a la antigua página con la nueva página creada. Esto empuja recursivamente todas las actualizaciones hacia niveles de jerarquía superior, hasta llegar a la raíz del FS, que a menudo se almacena en un registro muy pequeño.

Estos FSs son interesantes debido a que ofrecen un rendimiento muy similar a los FSs basados en páginas al lograr actualizaciones atómicas sin almacenar cambios de datos directamente en un registro. Incluso desvinculan la ubicación de almacenamiento de los datos, lo que crea una oportunidad para la nivelación de desgaste.

Sin embargo también presentan dificultades. Debido a que las actualizaciones en un FS basado en COW no se detienen hasta que alcanzan la raíz, una actualización puede desencadenar un conjunto más grande de escrituras de las necesarias para los datos originales. Este fenómeno es conocido como amplificación de escritura. Además, el movimiento jerárquicamente ascendente concentra estas escrituras en el bloque, que puede desgastarse mucho antes

que el resto de bloques del FS. Las figuras 1 y 2 ilustran el comportamiento de los datos y el almacenamiento al modificar un nodo en este tipo de FS.

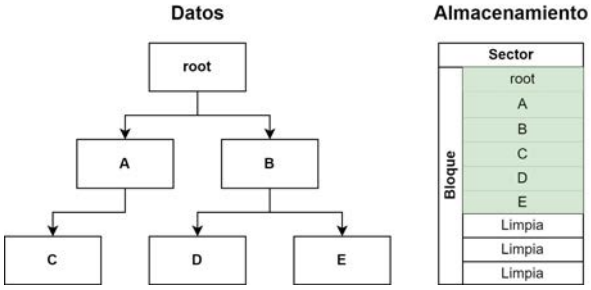

Figura 1: Estado antes de modificar el nodo D.

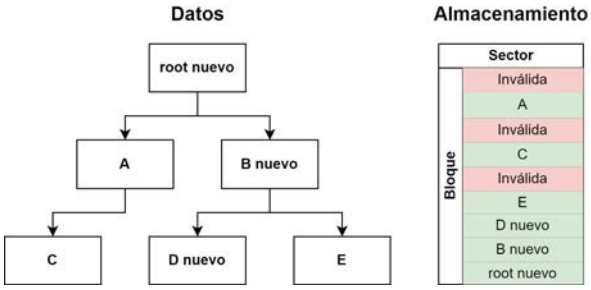

Figura 2: Estado luego de modificar el nodo D, que desencadena modificaciones en los nodos B y root.

I-B. Trabajo previo relacionado

En esta sección se puede encontrar un análisis del trabajo previo relacionado a diseños de algoritmos de nivelación de desgaste (WL) y recolector de basura (GC), detallado en las tablas I y II respectivamente. Vale la pena mencionar también el análisis realizado por Li-Pin Chang [1] ya que compara en detalle el rendimiento de múltiples tipos de algoritmos de nivelación de desgaste.

Tabla I: Trabajo relacionado a diseños de algoritmos de WL.

Algoritmo	Descripción	Problema
Encolamiento de páginas [1]	Utiliza dos colas para registrar las páginas con datos calientes y datos fríos, intercambiándolas entre sí cuando sea necesario.	Requiere gran cantidad de RAM para almacenar las referencias de todas las páginas.
Periódico [2]	Utiliza condiciones de disparo periódicas para verificar el desgaste de los sectores y guía las escrituras hacia los sectores menos desgastados.	Incrementan el tiempo ocupado para realizar WL, ejecutándose incluso en los momentos donde una WL no sea necesaria.
Según edad del sector [3] [4]	Divide a los sectores como ”viejos” o ”jóvenes” según sus recuentos de borrado y calcula el lapso de tiempo transcurrido desde la última modificación de un sector.	Complejidad en las fórmulas utilizadas, disminuyendo el rendimiento del FS.

II. Diseño e implementación

II-A. Recolector de basura

El recolector de basura (GC) es un subsistema inevitable de cualquier FS que implemente la política de COW, ya que es el encargado de reciclar el espacio de memoria invalidado para poder seguir escribiendo. Este proceso puede involucrar también escrituras adicionales por parte del GC al intentar reciclar sectores que no se encuentran invalidados en su totalidad, es decir que aún poseen algunas páginas válidas cuyos datos deben copiarse a otros sectores limpios antes de poder borrarse. Por este motivo es que también la actividad del GC degrada dramáticamente el rendimiento del FS de una manera completamente impredecible.

Tabla II: Trabajo relacionado a diseños de algoritmos de GC.

Algoritmo	Descripción	Problema
Codicioso (GR) [5]	Selecciona el sector con la menor cantidad de páginas válidas como sector víctima.	No considera la nivelación de desgaste. No funciona bien para accesos de memoria con alta localidad espacial de referencia.
Costo-Beneficio (CB) [6]	Calcula un valor de costo-beneficio para cada sector y selecciona el sector con el valor más alto como víctima.	Puede disparar recolecciones innecesarias sobre esquemas estáticos. No considera el recuento de borrados para cada sector.
Costo-Edad-Tiempo (CAT) [7]	Extiende el algoritmo CB al considerar el recuento de borrados para cada sector al seleccionar un sector víctima.	Acarrea un costo computacional periódico.
Activo de Tiempo Real [8]	Aprovecha el tiempo inactivo de la FTL entre solicitudes y lanza activamente tareas para recuperar espacio invalidado.	Basado en una FTL y completamente ligado a un RTOS que pueda administrar tareas.
Consciente del Archivo (FaGC) [9]	Copia páginas válidas en un sector víctima a clústeres en sectores libres según la frecuencia de actualización calculada del fragmento del archivo asociado.	Requiere gran cantidad de recursos para almacenamiento de tablas de traducción y registros temporales.

Se puede decir que la gestión de la sobrecarga del GC es un problema clave que debe tenerse en cuenta en la etapa inicial del diseño arquitectónico de un FS. Existen múltiples diseños de algoritmos de GC cuyas características dependen del tipo de FS, políticas y memoria sobre la cual están implementados. Estos diseños existentes se mencionan en la tabla II y se ha concluido en que ninguno de ellos se adapta completamente a los requerimientos de este trabajo.

En este trabajo se propone un nuevo algoritmo de GC para memorias flash NAND bajo la política de COW. El diseño de este GC se ilustra en la figura 3 y tiene por objetivo:

- Maximizar el rendimiento de la recolección de basura apoyándose también en el trabajo que realiza el algoritmo de WL.
- Minimizar el tiempo total de recolección de basura ocupado, aplicando una política de recolección a demanda.
- Contemplar todos los esquemas de memoria flash posibles relacionados con la recolección de basura a lo largo del ciclo de vida del FS.

Este algoritmo también se inspira y adapta de los conceptos y características con mayor aporte de valor de los trabajos relacionados mencionados en la tabla II, tales como:

- El concepto de identificación y selección de un sector como sector víctima, inspirado en el algoritmo GR,

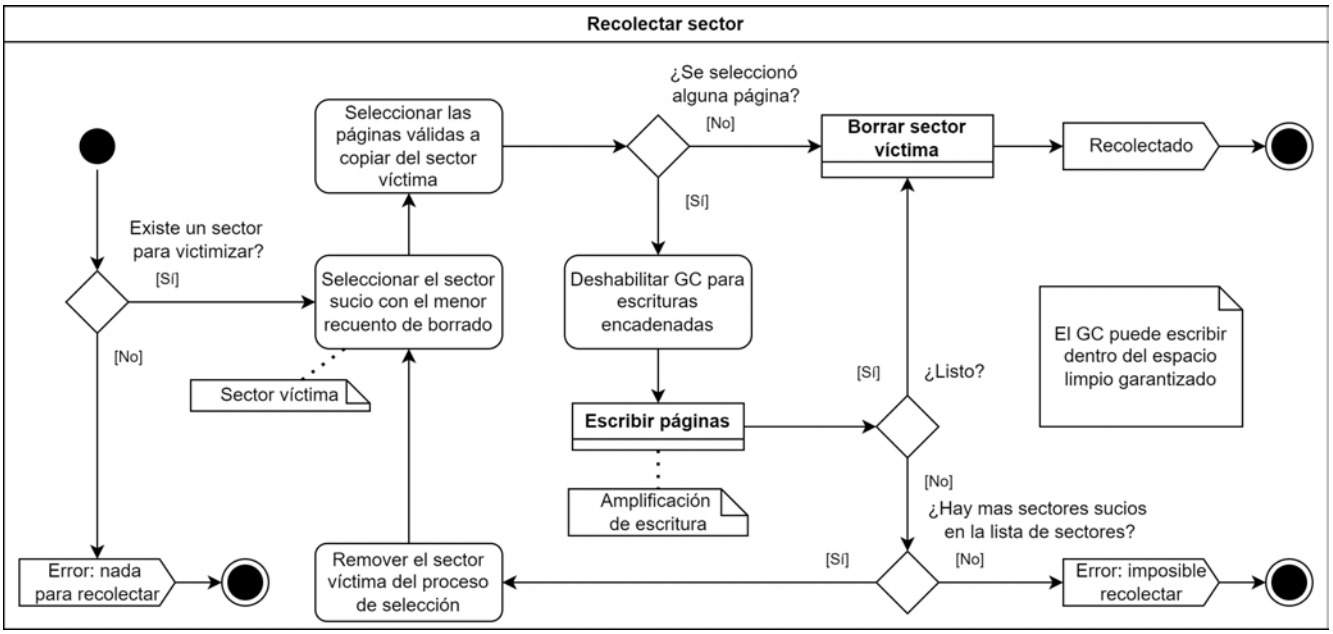

Figura 3: Algoritmo de GC.

con la diferencia en la lógica de selección ya que la víctima puede seleccionarse con al menos una página inválida y no necesariamente alguna página válida.

- El uso de listas de sectores para organizar los procesos a ejecutar, inspirado en el algoritmo activo de tiempo real, con la diferencia que se implementa una única lista y con una política de ordenamiento según el recuento de borrado.
- El uso de disparadores para la ejecución del GC a demanda, inspirado en el algoritmo FaGC, evitando el costo computacional de ejecuciones periódicas. Con la diferencia de que los criterios de disparos son completamente diferentes al no utilizar excesivos recursos en modificaciones y almacenamientos de tablas.

En el presente diseño, un ciclo de ejecución sin errores del GC realiza la recolección de únicamente un solo sector, denominado sector víctima. Un sector puede ser seleccionado como víctima por el GC si dicho sector es un sector sucio, es decir que contiene al menos una página inválida para recolectar. Cuando se invoca al GC, este busca dentro de la lista de sectores por el sector sucio con el menor recuento de borrado para victimizar. Se priorizan los sectores de menor recuento de borrado para favorecer al desgaste homogéneo de la memoria flash, ya que finalmente el GC borra el sector en caso de una ejecución sin errores. Si el GC no encuentra una víctima, significa que no existen sectores sucios (ergo páginas inválidas) en la memoria flash, entonces retorna un mensaje de error indicando que no hay ningún espacio inválido para recolectar.

Si el GC encuentra una víctima, entonces procede a realizar su recolección. La recolección solo es posible si es posible realizar COW de las páginas válidas del sector víctima sobre un sector objetivo, esto significa verificar que las páginas válidas del sector víctima puedan ser copiadas a páginas limpias de otros sectores en memoria flash. No se permite realizar COW de las páginas válidas del sector víctima sobre el mismo sector ya que finalmente el GC borra el sector víctima en caso de una ejecución sin errores, borrando las páginas válidas copiadas y corrompiendo el FS. Por este motivo, un sector no puede ser víctima y objetivo al mismo tiempo.

Cuando la recolección puede realizarse de manera exitosa, el GC selecciona las páginas válidas del sector víctima y las copia fuera del sector antes de ser borrado. Esta etapa adiciona procesos de escritura inevitables que contribuyen a la amplificación de escritura. Puede ocurrir también que el sector víctima no posea páginas válidas, en ese caso el GC evita ejecutar escrituras adicionales para proceder directamente a borrar el sector víctima antes de finalizar su ejecución indicando con una señal de éxito que pudo recolectar el sector.

El GC siempre realiza de manera exitosa la recolección de un sector víctima sobre el que se pueda realizar COW. De no poder realizar COW, entonces el GC itera sobre la lista de sectores en busca de otro sector a victimizar sobre el que se pueda realizar una recolección según la política COW y, de no encontrarlo, retorna un mensaje de error indicando que ninguna víctima pudo ser recolectada.

II-B. Nivelación de desgaste

Uno de los principales desafíos de la memoria flash NAND es la resistencia limitada de sus sectores a los ciclos de programación y borrado. Típicamente, la memoria flash NAND puede soportar 100.000 de estos ciclos para el tipo de celda de nivel único (SLC) o 10.000 para el tipo de celda de nivel múltiple (MLC). La barrera dieléctrica en las celdas de memoria flash se desgasta progresivamente debido a las operaciones de borrado. Cuando la barrera dieléctrica se rompe (se desgasta completamente), la celda de memoria flash ya no puede almacenar información de bits de manera confiable y se dice que la página es ahora una página mala.

Algunos sectores pueden desgastarse antes que otros debido a las operaciones excesivas de programación y borrado, reduciendo el tamaño efectivo de la memoria y, en

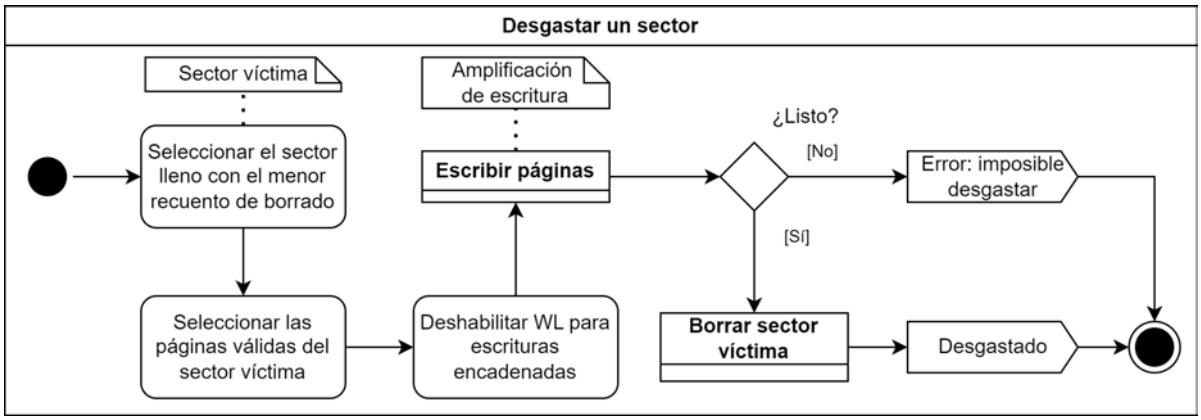

Figura 4: Algoritmo de WL.

consecuencia, afectando negativamente la vida útil de los dispositivos basados en memorias flash. Para resolver este problema, comúnmente se utilizan técnicas de nivelación de desgaste (WL) para mejorar la vida útil de la memoria flash NAND al igualar el ritmo de desgaste entre todos los sectores según un criterio de desgaste [10] [11].

El objetivo de la nivelación de desgaste es distribuir de manera uniforme el ritmo de desgaste entre todos los sectores y así evitar que algunos sectores se desgasten más rápido que otros, lo que puede maximizar la resistencia y la vida útil general de la memoria flash NAND. Actualmente, los esquemas de WL existentes se pueden dividir en dos categorías según el tipo de datos que contemplan: nivelación de desgaste dinámica (DWL) y nivelación de desgaste estática (SWL). Los diseños de DWL reciclan sectores según su recuento de borrado absoluto y solo afectan a los datos dinámicos, también llamados datos calientes, que cambian con frecuencia. Distintos de los diseños de DWL, los diseños de SWL tienen en cuenta las características de acceso a los datos y contemplan también los datos estáticos, también llamados datos fríos. La SWL mueve los datos fríos a los sectores con mayor desgaste, a cambio de introducir una sobrecarga adicional en el rendimiento y en el código.

Teniendo en cuenta que el FS del presente trabajo almacenará, entre otras cosas, archivos de configuración y calibración del dispositivo embebido, y que dichos archivos son de carácter estático, se considera que la categoría adecuada de WL a implementar es la SWL, por lo que los análisis siguientes se desarrollaran únicamente para esta categoría. Los diseños existentes de algoritmos de SWL se mencionan en la tabla I y se ha concluido en que ninguno de ellos se adapta completamente a los requerimientos de este trabajo.

En este trabajo se propone un nuevo algoritmo de WL para memorias flash NAND, cuyo diseño se observa en la figura 4 y tiene por objetivo:

- Maximizar el rendimiento de la nivelación de desgaste apoyándose también en el trabajo que realiza el algoritmo del GC.
- Minimizar el tiempo total de nivelación de desgaste ocupado, aplicando una política de nivelación a demanda.
- Contemplar todos los esquemas de memoria flash posibles relacionados con la nivelación de desgaste a lo largo del ciclo de vida del FS.

Este algoritmo también se inspira y adapta de los conceptos y características con mayor aporte de valor de los trabajos relacionado mencionados en la tabla I, tales como:

- El enfoque en los recuentos de borrado de los sectores para la identificación de datos fríos y calientes, inspirado en el algoritmo periódico, con la diferencia en el reemplazo de la política de ejecución periódica por una política de ejecución a demanda.
- La verificación en tiempo de escritura y el uso de colas para la administración y manejo de los datos, inspirado en el algoritmo de encolamiento de páginas. Con la diferencia en la utilización de una única cola a nivel de sectores en vez de a nivel de páginas, lo que reduce drásticamente el tamaño y la longitud de la cola, permitiendo amortizar el tiempo de procesamiento de los algoritmos de búsqueda de complejidad $O(n)$.

El presente diseño de algoritmo entra en la categoría de SWL. Este algoritmo de WL es invocado por el FS, bajo una política de ejecución a demanda, cuando el FS intenta escribir en un sector cuya diferencia de recuento de borrado respecto al sector de menor recuento de borrado en la memoria flash es superior a un umbral preestablecido. Al ejecutarse, el algoritmo de WL mueve los datos del sector de menor recuento de borrado (el menos desgastado) hacia los sectores de mayor recuento de borrado (los más desgastados) para posteriormente borrar y reutilizar el sector menos desgastado.

En el momento en el que el FS invoca al algoritmo de WL, se pueden hacer las siguientes asunciones respecto al esquema de memoria, garantizadas por diseño por contrato (DBC).

1. El sector sobre el cual el FS intentó escribir es un sector disponible cuya diferencia de recuento de borrado respecto al sector de menor recuento de borrado es superior al umbral de WL preestablecido.
2. El sector de menor recuento de borrado en la memoria flash es un sector lleno. Esto se debe a que si el sector no estuviera lleno, entonces el FS hubiera intentando escribir sobre dicho sector hasta llenarlo, ya que el FS siempre prioriza escribir datos sobre el sector de menor recuento de borrado disponible.

Cuando se ejecuta el algoritmo de WL, el mismo busca en

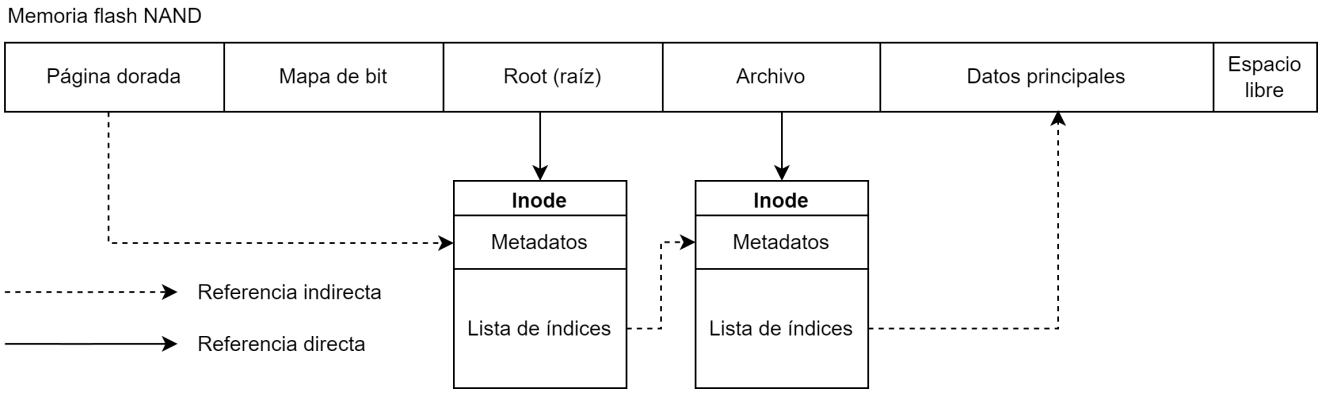

Figura 5: Arquitectura general del FS.

la lista de sectores por un sector a victimizar sobre el cual realizar un desgaste, es decir una operación de borrado. El sector víctima a seleccionar es aquel sector lleno que posea el menor recuento de borrado. Se considera que este sector víctima posee datos fríos o estáticos que deben moverse a otro sector debido a que estos no se han modificado con la misma frecuencia que los datos calientes o dinámicos en el resto de la memoria flash que contribuyen a un desnivel en el desgaste de los sectores de la memoria.

Los datos que se mueven desde el sector menos desgastado (frío) hacia el sector más desgastado (caliente) son únicamente los datos válidos del sector. Si el sector menos desgastado posee un espacio inválido, este será limpiado automáticamente al borrar el sector al final del proceso, contribuyendo al proceso de recolección de basura.

Al igual que el GC, el algoritmo de WL solo puede mover los datos válidos si puede realizar COW exitosamente, y en caso de no poder realizarlo retorna un mensaje de error indicando que el sector víctima no pudo ser desgastado.

II-C. Formateo y montado

Los procesos de formateo y montado incluyen estructurar la memoria flash según la arquitectura del FS e identificar luego dicha arquitectura, respectivamente. La figura 5 ilustra la arquitectura general diseñada para el presente FS, que cuenta con las siguientes áreas principales:

- Página dorada: también conocida popularmente como super bloque, es una página especial que contiene la metadata más importante del FS, i.e.: las referencias al directorio raíz y al mapa de bits. Un proceso de montado exitoso es aquel en el que se identifica la página dorada en memoria flash y puede cargarse su metadata a memoria volátil para ser utilizada posteriormente.
- Mapa de bit: es una representación instantánea del estado de validez de todas las páginas en memoria flash. Este mapa se actualiza junto a cada actualización de datos exitosa en la memoria flash y sirve como punto de control para el FS. Ante una posible pérdida de energía que interrumpa un proceso de actualización de datos, el FS puede identificar si un conjunto de páginas válidas pertenece al punto de control actual y, de no hacerlo, invalidar las mismas.
- Directorio raíz y archivos: todos los FS analizados cuentan con un directorio raíz, y el FS del presente trabajo no es la excepción. Los directorios y archivos se almacenan como nodos de índices que contienen referencias indirectas a otras estructuras de datos. En el caso de directorios, cada índice representa un archivo. En el caso de archivos, cada índice representa una página de sus datos.

A cada página en memoria flash, dentro del espacio asignado al FS, se le almacena un encabezado que contiene los metadatos esenciales para ser operada por el FS, ilustrado en la figura 6. El encabezado tiene una longitud de 12 bytes y contiene los siguientes campos de información:

- Recuento de borrado: indica el recuento de borrado del sector al cual pertenece la página.
- Longitud de datos: indica la longitud de los datos válidos dentro de la página, sin tener en cuenta el propio encabezado.
- Estado: indica el estado de la página, i.e.: limpia, válida, inválida o mala.
- Tipo: indica el tipo de datos almacenados en la página, i.e.: de página dorada, de mapa de bits, de directorio raíz, de archivo o de datos principales.
- Identificador de nodo: indica el identificador único del nodo de un archivo.
- Índice de nodo: indica a qué índice, de un nodo padre, pertenece la página.

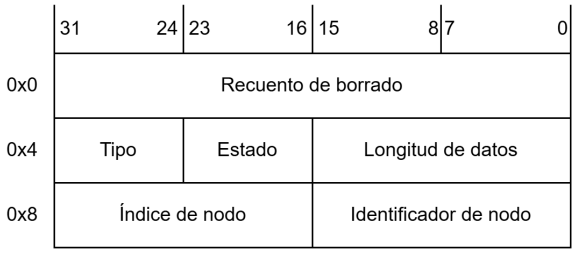

Figura 6: Encabezado de página.

Durante el formateo, se borra completamente el espacio asignado al FS en memoria flash y se escriben los encabezados de páginas. Luego se escriben el mapa de bits, el directorio raíz vacío y la página dorada con referencias a las dos áreas previas. En cada escritura de página se actualiza su encabezado con los metadatos correspondientes.

Durante el montado, el FS realiza un escaneo de las páginas válidas de memoria flash en búsqueda de un encabezado

correspondiente a la página dorada. Una vez encontrada, su contenido se copia a estructuras en memoria volátil, incluyendo el mapa de bits y la referencia al directorio raíz. Si no se encuentra la página dorada, el proceso de montado falla y retorna un mensaje de error.

III. Ensayos y resultados

El ambiente de ensayo utilizado consiste en un esquema de conexión trivial al trabajar con placas de desarrollo, conformado por tres elementos principales: una computadora u ordenador, un microcontrolador (MCU) y una sonda de depuración, que normalmente se encuentra integrada en la placa de desarrollo junto a otros periféricos que facilitan el acceso al microcontrolador.

El presente trabajo, además, necesita de una memoria a la que poder administrar desde el firmware del microcontrolador. La memoria es suministrada por la placa de desarrollo misma, STM32 Nucleo-144, que cuenta con una memoria flash embebida.

La figura 7 ilustra la conexión entre la computadora y la placa de desarrollo. La computadora se comunica con el microcontrolador a través de una sonda de depuración y recibe mensajes de este sobre un puerto serie dedicado. El firmware del microcontrolador, que contiene una aplicación para evaluar el presente FS, se comunica con la memoria flash embebida para administrar la información de la aplicación.

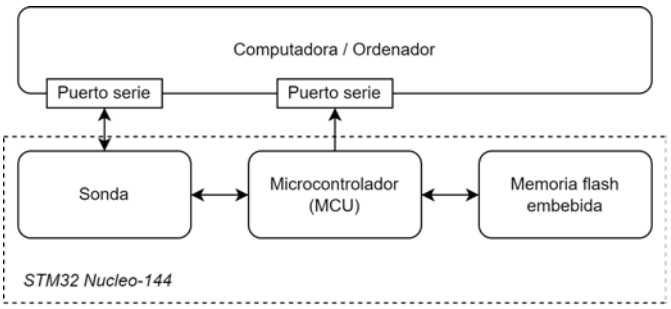

Figura 7: Diagrama en bloques de conexiones del ambiente de ensayo.

III-A. Rutina de firmware del microcontrolador

La rutina de firmware del microcontrolador consiste en dos etapas. En la primera etapa, se inicializan los periféricos necesarios y se realiza el montaje y/o formateo del FS. En la segunda etapa, el firmware queda a la espera, de manera indefinida y parpadeando un LED como señal de vida, de recibir dos eventos: uno periódico y otro aperiódico.

El evento periódico es enviado por la rutina misma, y dispara una lectura de archivo en memoria flash. El archivo posee la información necesaria para indicarle a la aplicación que recurso de hardware disponible debe utilizar. En este caso le indica cuál LED en la placa de desarrollo debe encender.

El evento aperiódico es enviado por el usuario mediante un pulsador en la placa de desarrollo. Este evento dispara la modificación del contenido del archivo de configuración en memoria flash, y modifica las especificaciones de los recursos de hardware a utilizar por la aplicación. En este caso indicándole cuál otro LED debe ser utilizado.

Además, el progreso de ambas etapas es reportado a un puerto serie para complementar los ensayos y la depuración.

III-B. Técnica de diseño de prueba

La técnica utilizada fue *Control Flow Test (CFT)*. Su objetivo es ensayar la estructura del programa. Cada ensayo o prueba consiste en un grupo de acciones que cubren un camino determinado a lo largo de un algoritmo o del programa. Esta es una técnica de diseño de prueba formal mayormente utilizada en pruebas unitarias y de integración.

La figura 8 ilustra el diagrama de flujo que representa el ensayo característico del presente FS. Cada punto de decisión y acción en el diagrama de flujo es identificado con una etiqueta única del tipo *DPx* y *ACx* respectivamente. Las acciones que se encuentren entre dos puntos de decisión sin bifurcaciones son consideradas como una acción conjunta y no son etiquetadas aunque se ilustren separadas para comprender el detalle individual de cada una de ellas.

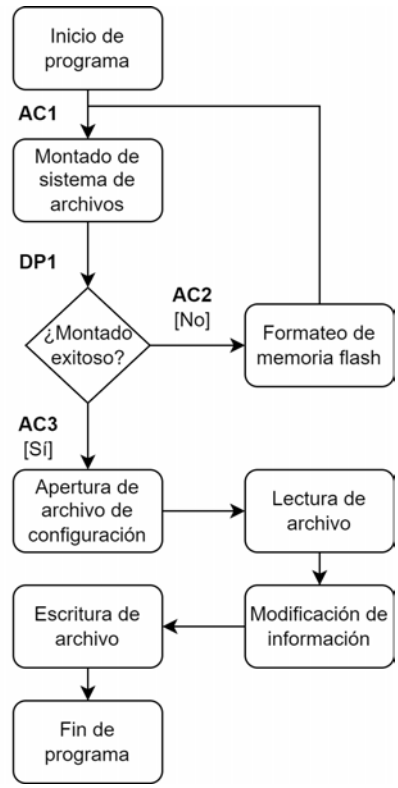

Figura 8: Diagrama de flujo de ensayo del sistema de archivos.

El nivel de profundidad de prueba utilizado para el PAB fue de 1 debido a su representación trivial. La combinación de acciones para los puntos de decisión son:

- - : (AC1)
- DP1 : (AC2); (AC3)

Esto lleva a la siguiente combinación de caminos:

- Camino 1: (AC1, AC3)
- Camino 2: (AC1, AC2, AC3)

III-C. Resultados

La figura 9 ilustra los resultados del ensayo realizado en memoria flash embebida. Se utilizaron todos los sectores de 16 KiB del segundo banco de memoria de la placa de

desarrollo. En la figura 9 se observan los mensajes recibidos por puerto serie durante varias ejecuciones del ensayo, con la arquitectura del FS y el archivo de configuración previamente almacenados en memoria flash. También, en color rojo, se indican los eventos externos que ocurrieron antes de recibir cada mensaje, cuando corresponda. Se observa entonces que:

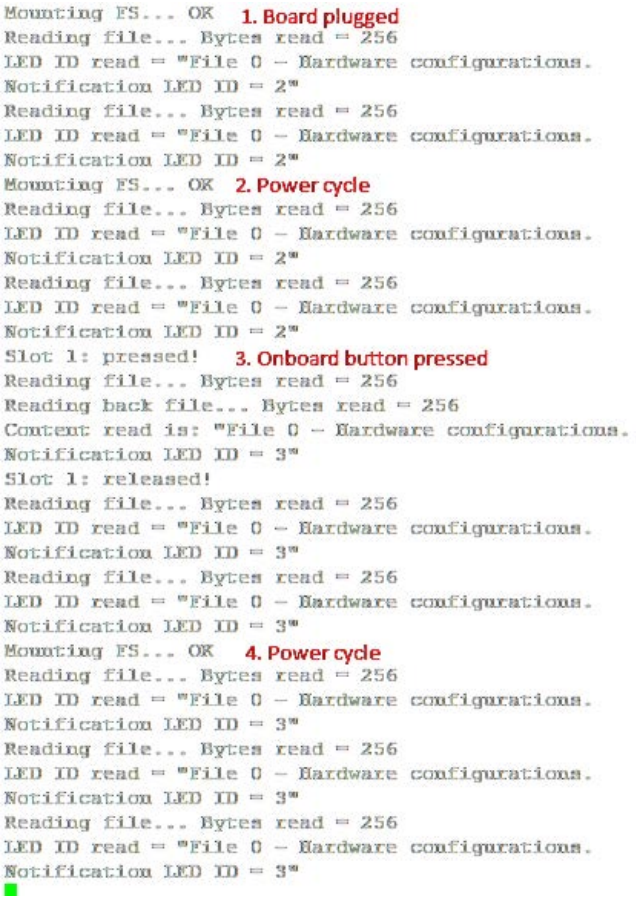

Figura 9: Ensayo en memoria no volátil. Mensajes recibidos por puerto serie.

1. El primer intento de montado del FS se resuelve de manera exitosa.
2. Se lee, modifica y guarda un archivo de configuración.
3. Se lee periódicamente el archivo de configuración y se valida que su contenido persiste luego de reiniciar la placa de desarrollo.

Finalmente se destaca el gran rendimiento alcanzado, en relación al uso general de la memoria flash y la sobrecarga de código aportada, que se menciona a continuación:

- El FS ocupó un espacio total de 3.084 bytes en ROM.
- El FS tuvo un consumo máximo de 340 bytes en RAM.
- El FS puede indexar hasta 2^{32} páginas.
- El FS ocupó, con metadatos, el 4,7 % del espacio útil de páginas para la administración de estas.

IV. Conclusión

Este trabajo presenta un sistema de archivos para memorias flash embebidas del tipo NAND, optimizado para aportar mínima sobrecarga de código. El trabajo aborda, de manera integral, diversos aspectos críticos relacionados con la administración eficiente de la memoria, el manejo de errores y la preservación de la integridad de los datos.

A lo largo de este documento se hizo énfasis en la importancia de proponer nuevos diseños para los subsistemas del trabajo y la necesidad de innovar sobre ellos. En consecuencia, se desarrollaron nuevos subsistemas principales de recolección de basura y nivelación de desgaste, entre otros, con enfoque en la técnica de copia al escribir para asegurar, de manera natural, la resistencia del sistema a la pérdida de energía.

El trabajo presenta una solución robusta, innovadora y adaptativa para los desafíos específicos de esta tecnología. La implementación de técnicas avanzadas y la consideración de aspectos como la amplificación de escritura y la integridad de los datos consolidan un sistema de archivos que no solo mejora el rendimiento y la longevidad de la memoria flash, sino que también proporciona una base sólida para futuras aplicaciones en almacenamiento crítico y sistemas embebidos.

Referencias

[1] L.-P. Chang, “On efficient wear leveling for large-scale flash-memory storage systems,” in *Proceedings of the 2007 ACM Symposium on Applied Computing*, ser. SAC ’07. New York, NY, USA: Association for Computing Machinery, 2007, p. 1126–1130. [Online]. Available: https://doi.org/10.1145/1244002.1244248

[2] L.-P. Chang and T.-W. Kuo, “Efficient management for large-scale flash-memory storage systems with resource conservation,” *ACM Trans. Storage*, vol. 1, no. 4, p. 381–418, Nov. 2005. [Online]. Available: https://doi.org/10.1145/1111609.1111610

[3] S. G. L. H. J. Kim, “An effective flash memory manager for reliable flash memory space management,” *IEICE Transactions on Information and System*, vol. E85-D, no. 6, pp. 950–964, 2002.

[4] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang, “Using data clustering to improve cleaning performance for flash memory,” *Software: Practice and Experience*, vol. 29, no. 3, pp. 267–290, 1999.

[5] M. Wu and W. Zwaenepoel, “envy: a nonvolatile main memory storage system,” in *Proceedings of IEEE 4th Workshop on Workstation Operating Systems. WWOS-III*, 1993, pp. 116–118.

[6] A. Kawaguchi, S. Nishioka, and H. Motoda, “A Flash-Memory based file system,” in *USENIX 1995 Technical Conference (USENIX 1995 Technical Conference)*. New Orleans, LA: USENIX Association, Jan. 1995. [Online]. Available: https://www.usenix.org/conference/usenix-1995-technical-conference/flash-memory-based-file-system

[7] M.-L. Chiang and R.-C. Chang, “Cleaning policies in mobile computers using flash memory,” *J. Syst. Softw.*, vol. 48, pp. 213–231, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:17022203

[8] S.-J. Syu and J. Chen, “An active space recycling mechanism for flash storage systems in real-time application environment,” in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA’05)*, 2005, pp. 53–59.

[9] H. Yan and Q. Yao, “An efficient file-aware garbage collection algorithm for nand flash-based consumer electronics,” *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 623–627, 2014.

[10] W. S. E., “Method for wear leveling in a flash eeprom memory,” *U.S. Patent and Trademark Office.*, no. (U.S. Patent No. 5,341,339), 1993.

[11] B. A., “Wear leveling of static areas in flash memory,” *U.S. Patent and Trademark Office.*, no. (U.S. Patent No. 6,732,221), 2001.