(12) UK Patent (19) GB (11) 2541916 (13) B

(45) Date of B Publication 09.05.2018

(54) Title of the Invention: Method of operating data memory and device utilizing method

(51) INT CL: ***G06F 3/06*** (2006.01)

(21) Application No: **1515658.1**

(22) Date of Filing: **03.09.2015**

(43) Date of A Publication **08.03.2017**

(56) Documents Cited:

US 5454103 A	**US 5375233 A**
US 20130111182 A1	**US 20110106806 A1**
US 20110040795 A1	**US 20110035557 A1**
US 20090112951 A1	**US 20050160309 A1**

(58) Field of Search:
As for published application 2541916 A viz:
INT CL **G06F**
Other: **WPI, EPODOC, TXTA**
updated as appropriate

(72) Inventor(s):
Tuomas Kärkkäinen
Ossi Mikael Kalevo

(73) Proprietor(s):
Gurulogic Microsystems Oy
Linnankatu 34, Turku FI-20100, Finland

(74) Agent and/or Address for Service:
Basck Ltd
16 Saxon Road, CAMBRIDGE, Cambridgeshire, CB5 8HS, United Kingdom

GB 2541916 B

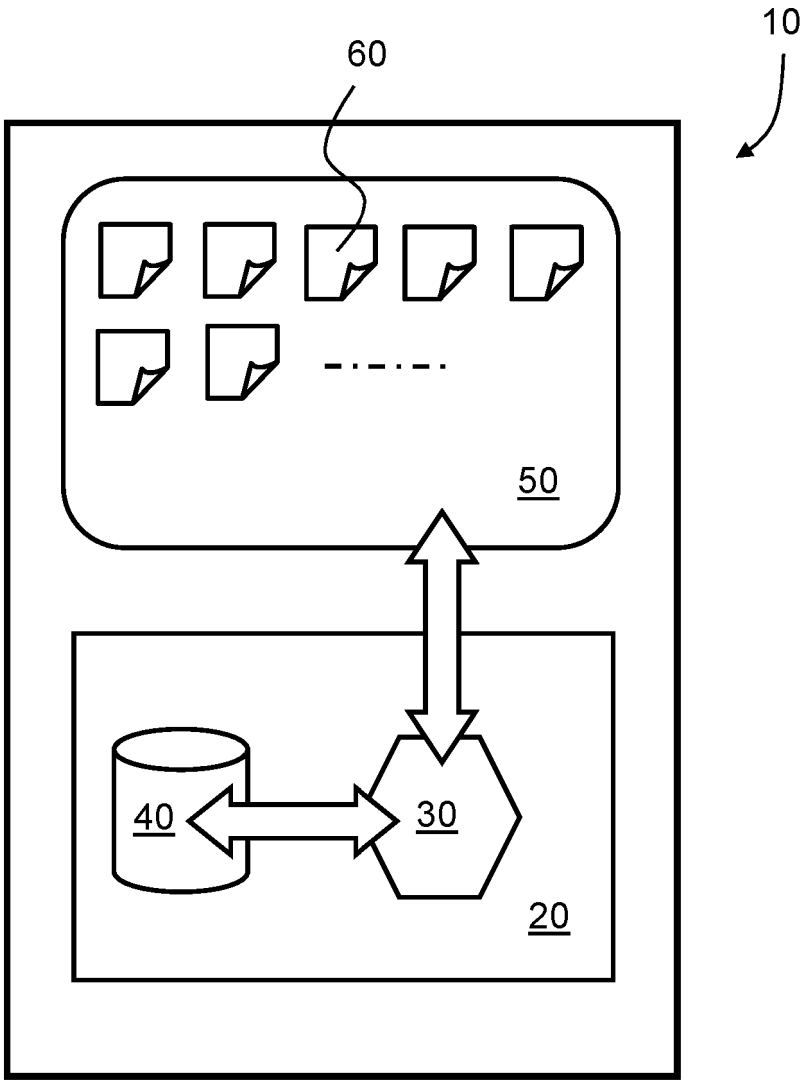

FIG. 1

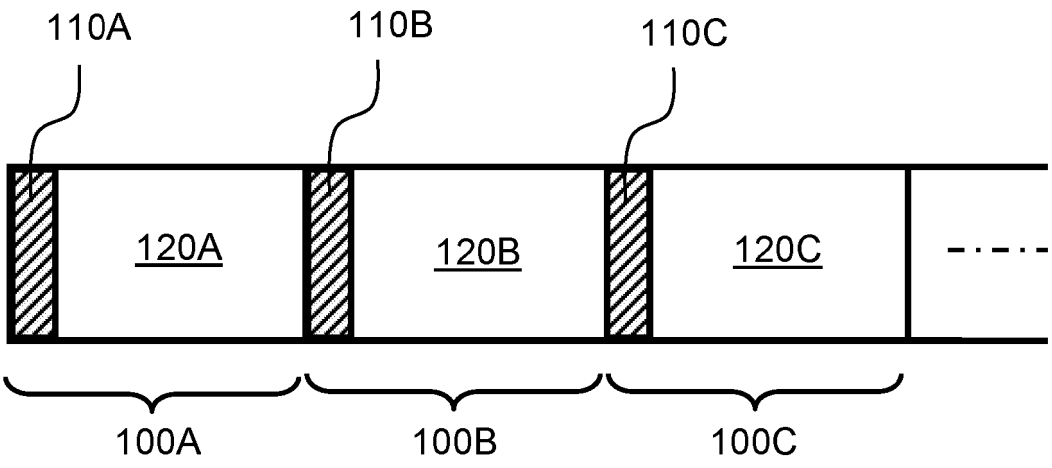

FIG. 2

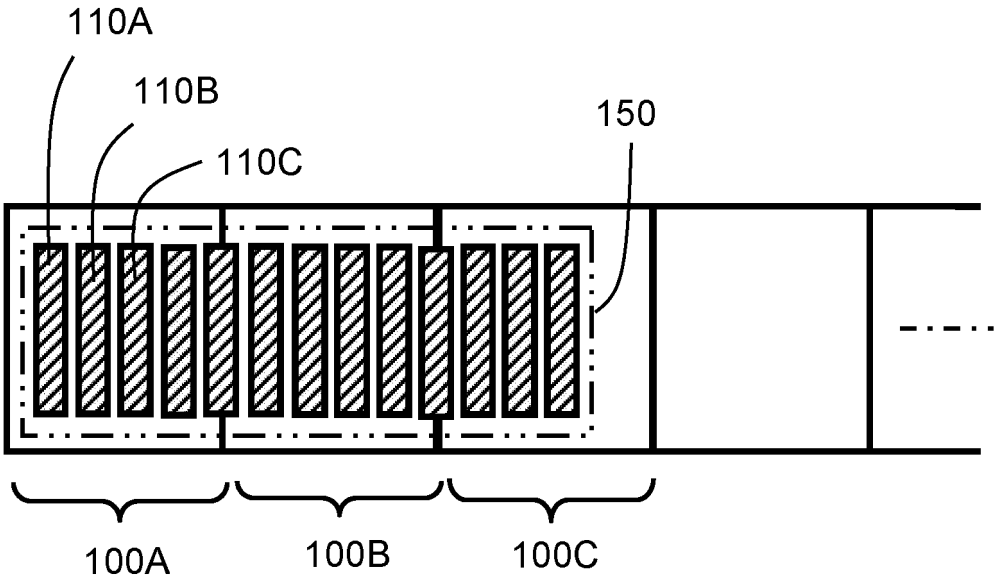

FIG. 3A

30 10 17

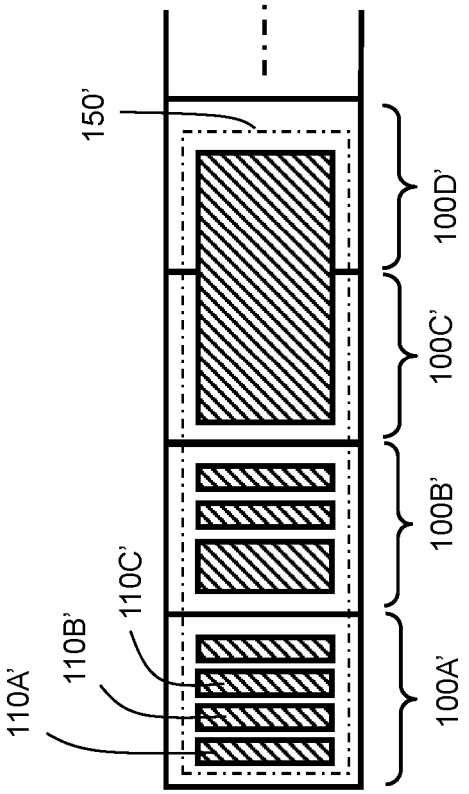

FIG. 3B

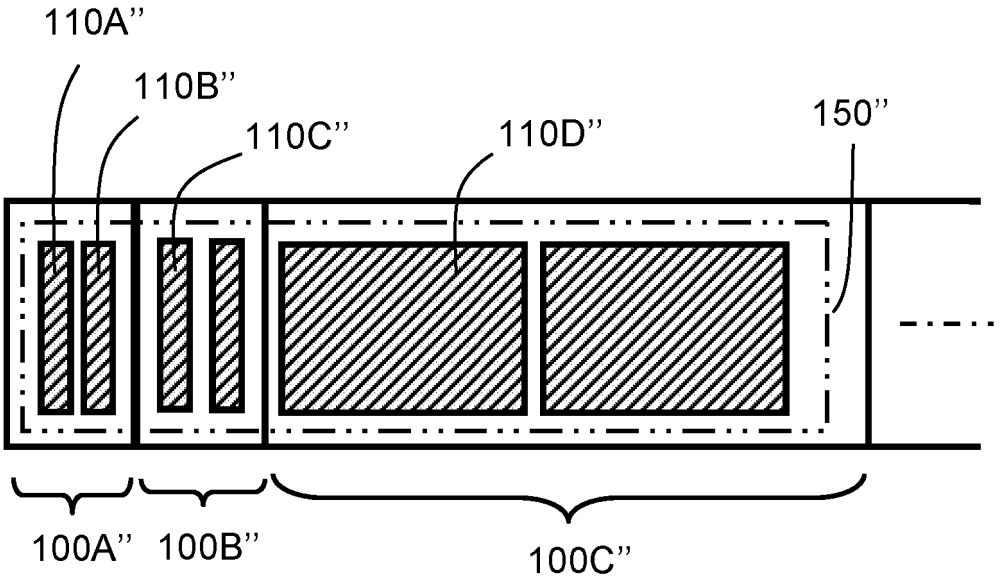

FIG. 3C

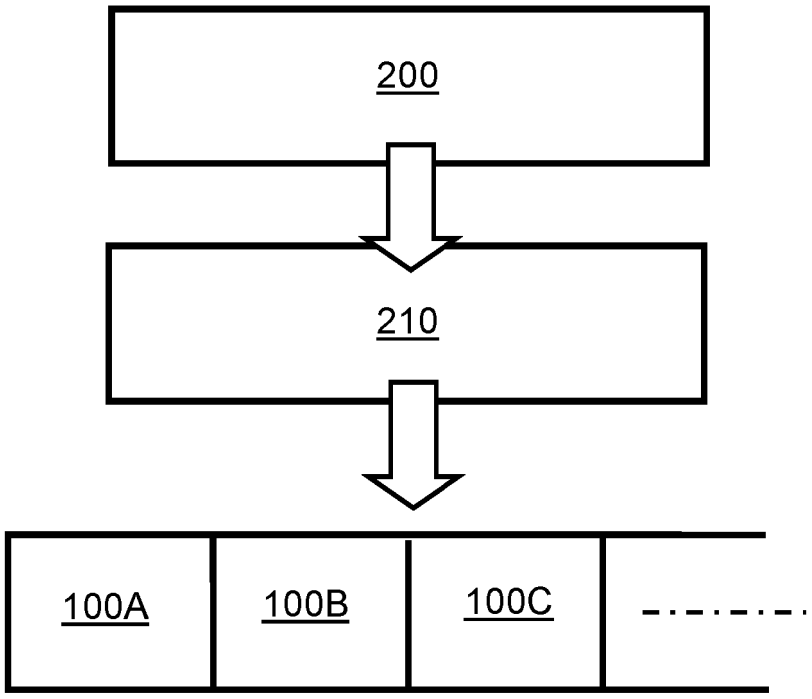

FIG. 4

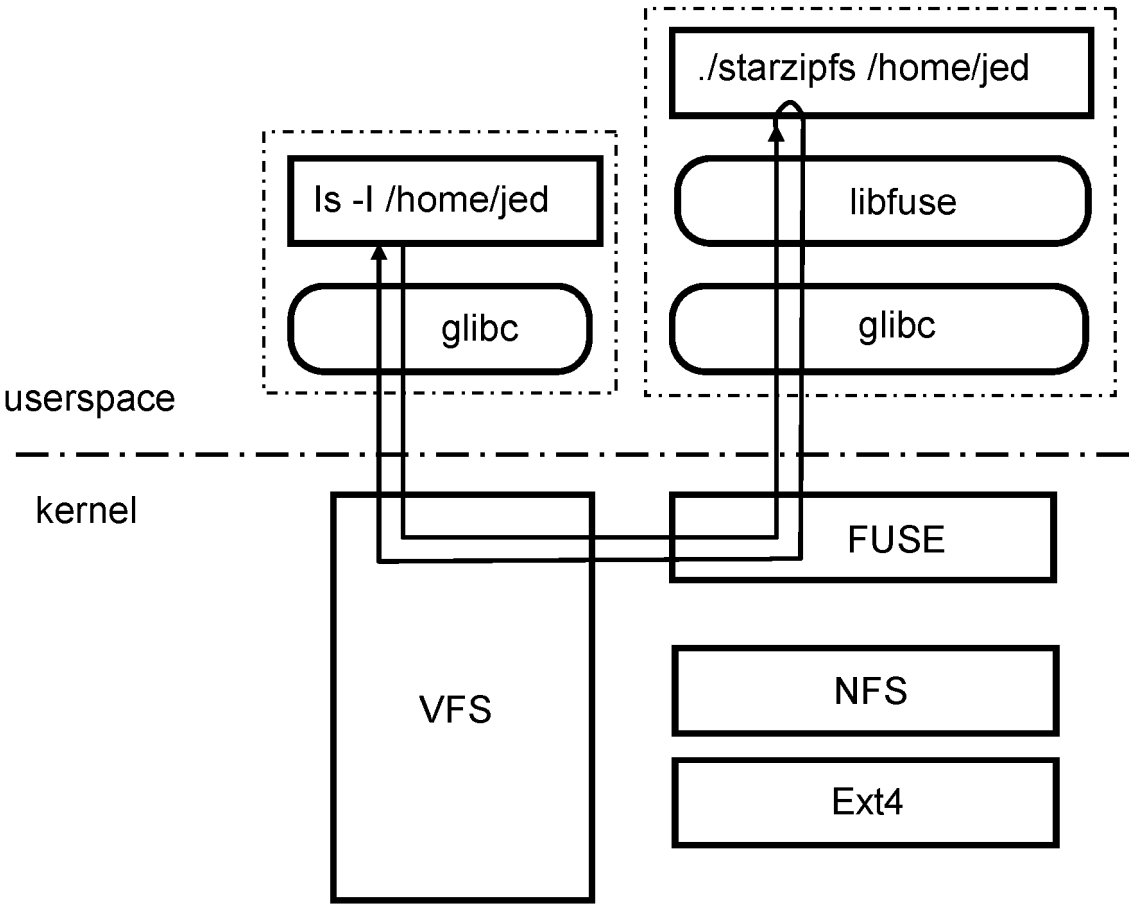
./starzipfs /home/jed
ls -l /home/jed
libfuse
glibc
glibc
userspace
kernel
FUSE
VFS
NFS
Ext4

FIG. 5a

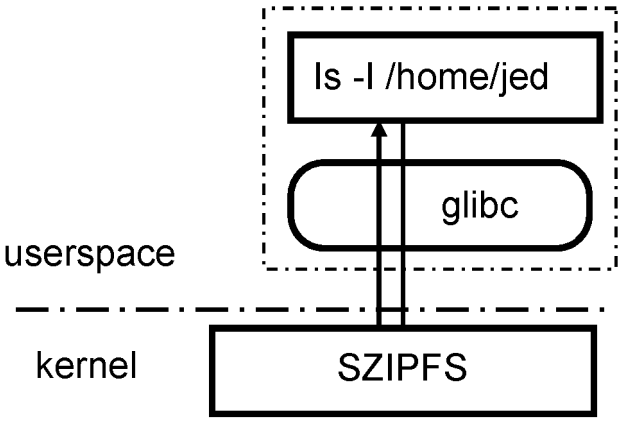
ls -l /home/jed
glibc
userspace
kernel
SZIPFS

FIG. 5b

METHOD OF OPERATING DATA MEMORY AND DEVICE UTILIZING METHOD

Technical Field

The present disclosure relates to methods of operating data memory, and also to devices which utilize the methods. Moreover, the present disclosure is concerned with computer program products comprising a non-transitory computer-readable storage medium having computer-readable instructions stored thereon, the computer-readable instructions being executable by a computerized device comprising processing hardware to execute aforesaid methods.

Background

Conventionally, portable electronic devices including one or more processors, data memory and a graphical user interface (GUI), for example games terminals and smart phones, employ icons to represent different functional options that are available to a user. Such icons, by way of their visual appearance, identify their corresponding functions. For example, the icons correspond to software applications, namely "Apps", which have been downloaded to the portable electronic devices.

A term "*slack space*" has commonly been used to refer to an unused memory space caused by files that are smaller than a cluster (namely a "*data cluster*") or a block (namely a block as a physical record) determined in a file system of the aforementioned portable electronic devices. Such slack space as a technical problem has however not been considered bad enough to warrant changes, because the average file size in such a file system has been considerably larger, and the price of non-volatile memory has previously not been significant in the production costs of the portable electronic devices. However, recently, the situation has changed as regards integrated systems and new non-mechanical, electrically-addressed ROM memories that are considerably more expensive than mechanically-addressed memories. Therefore, the technical problem has not been recognized earlier and thus there has hitherto been no need to find a technical solution for the technical problem. Moreover, along with the development of better compression methods, such as proprietary GMVC® associated with Gurulogic Microsystems Oy, Finland, the file sizes have become increasingly smaller, which in turn has increased the slack space.

Thus, there is an increased need to address such inefficiencies related to storage of data.

Gurulogic Microsystems®, namely a trademark of Gurulogic Microsystems Oy, Finland has previously invented a method of communicating data, as described in patent application GB 1504336.7, reference [3], namely "*Method of Communicating Data Packets within Data Communication Systems*"; the method produces optimal data packets for transfer in networks and file systems, thereby achieving a very cost-efficient data transfer mechanism that yields an almost theoretically maximal transfer capacity of information. It is used to combine different types of data together based on their priorities, and it offers a better user experience than known prior art techniques, especially in interactive communication systems.

Another known example of data compression concerns archives such as TAR or ZIP, which both are able to archive, and if necessary to compress, a large number of files into one physical file that thus comprises an optimal data container; however, there is a technical disadvantage that files inside the archive cannot be used individually, directly without extracting the files physically into another location.

It is previously known that contemporary database systems such as Oracle, MS-SQL, MySQL and MariaDB function as highly advanced and optimal data containers; "*Oracle*" is a registered trademark. It is also known that their technical implementation enables a cost-efficient solution to be achieved; databases store files in the binary (BLOB) format into a database table, which usually together comprise a large physical file.

One example of such is the software application Dropbox, which integrates itself into an existing file system, thereby enabling use of files directly as if they would physically exist in the existing file system; "*Dropbox*" is a registered trademark. Such an approach enables an optimized data container to be achieved, except that the Dropbox-style service operates in the cloud, and is not suitable for use in a local manner in portable electronic devices. "*Cloud*" refers to data storage capacity available in a data communication network, for example in the contemporary Internet, even if users are unaware precisely where in the data communication network their

data is stored; such cloud-based data storage usually occurs at one or more servers located in the data communication network.

Another known virtual file system based on use of a single file, described in reference [7], operates by having its files located in a virtual drive, wherein the files are directly usable individually without extraction or any additional copying into a physical location. However, even the virtual file systems are formatted into clusters (data clusters) or blocks (blocks as physical record) of pre-determined fixed size, and thus they still cause a slack space problem described and addressed in the present disclosure.

When a portable electronic device employs clusters as a minimum file system storage unit, use of sophisticated optimized data compression results in data, for example corresponding to GUI icons, employing even less space in memory, resulting in use of the clusters of the file system being even more inefficient.

It will be appreciated that files that are larger than the size of a cluster (data cluster) or block (block as physical record) used in the given data filing system cause slack space, because the actual content of a given file is rarely equal to the last cluster (data cluster) or block (block as physical record) reserved for the given file; in other words, files that are larger than the size of a cluster are rarely precisely of a size which is exactly an integer multiple of the size of the cluster. However, the more clusters (data clusters) or blocks (blocks as physical record) that are present in the file, the relatively less physical slack space is created in the data filing system when storing data therein.

In a published United States patent document US20110035557A1 (Olli Olavi Luukkainen; "*Fragmentation reduction using virtual sectors for static data*"; assigned to Nokia Corporation), there is described a method for facilitating enhanced storage efficiency when storing data. The method comprises:

(a) identifying a requirement for accessing at least one read/write block in a memory, wherein the read/write block is identified by a range of logical sector addresses;

(b) determining whether each logical sector address in the range of logical sector addresses corresponds to a physical sector or a virtual sector;

(c) if the address corresponds to a physical sector, determining the location of the physical sector and accessing the physical sector; and

(d) if the address corresponds to a virtual sector, performing an alternative activity as defined by an intermediate control layer.

In another published United States patent document US20110040795A1 (Colin Stebbins Gordon *et al.*; *"Merging containers in a multi-container system"*), there is described an apparatus and method for merging data of a first container and a second container into a target container that includes at least one block having a plurality of extents to store data objects. The apparatus includes a storage sever coupled to a plurality of storage devices. The storage server is configured to merge multiple data objects of the plurality of containers that store a data object per each container into a target container that stores multiple data objects within the target container.

In another published United States patent document US20130111182A1 (Vishal Chittranjan Aslot *et al.*, "*"Storing a small file with a reduced storage and memory footprint"*, assigned to International Business Machines Corporation), there is described a method for storing files, the method comprising:

(a) a processor receiving a first I/O request to store a first file in a file-system of a data storage device;

(b) the processor determining whether the size of the first file does not exceed a threshold size, wherein exceeding the threshold results in storing at least a portion of the first file in a block of the file-system devoid of sub-blocks;

(c) the processor determining whether the size of the first file does not exceed a size of unallocated space within a single block in the file system, the single block comprising a set of sub-blocks; and

(d) the processor, responsive to determining that the size of the first file does not exceed the threshold size, and responsive to determining that the size of the first file does not exceed the size of unallocated space within the single block in the file-system, storing, at a first address, the first file in a first subset of the set

of the sub-blocks of the single block; wherein the first address identifies the single block and a position of a sub-block in the subset.

In another published United States patent document US20090112951A1 (Seung-Woo Ryu *et al.*, "*Apparatus and method of managing files and memory device*"), there is described a system and method of managing files and a memory device. The system for managing files includes an attribute-management module designating nested attributes to the files having a size smaller than a basic allocation unit of a file system or to directories to which the files belong, a data-management module adjacently nesting the files to which the nested attributes are designated by the attribute-management module with no empty space between the files, and a metadata-management module writing start offsets of the files nested by the data-management module in metadata of the files.

In another published United States patent document US20110106806A1 (Alexis Tamas *et al.*, "*Process for optimizing file storage systems*", assigned to STG Interactive), there is described a system which includes a selection module, a file module, a storage cache, and an access module. The selection module organizes small files into groups according to a selection function, which organizes the small files based on at least one of related content of the small files, related filenames of the small files, and related access patterns of the small files. The file module uses a predetermined block size and stores, for each group, a large file containing all the small files of the group. The access module receives an access request for one of the small files from a client device and determines a large file corresponding to the one of the small files based on input from the selection module. The access module selectively reads the corresponding large file from the file module into the storage cache, and accesses the one of the small files from the large file.

In another published United States patent document US20050160309A1 (Richard Golding; "*Grouped-object RAID*"; assigned to International Business Machines Corp), there is described a RAID-configured grouped-object storage system which provides reduced storage space overhead for small objects. The storage system includes a plurality stripes arranged across a plurality of physical objects. Each stripe includes a plurality of storage blocks that are each mapped on to a respectively different

physical object. The storage system also includes a plurality of virtual objects each containing at least one storage block. A group of virtual objects is formed when a virtual object contains less storage blocks than the number of stripes by associating the virtual object with at least one virtual object containing less storage blocks than the number of stripes and/or at least one storage block containing zero values so that the storage blocks of each group of virtual objects equals the number of stripes. The storage blocks of each virtual object and of each group of virtual objects are mapped to a respectively different stripe.

Summary

The present disclosure seeks to provide a more efficient method of storing data, for example data content objects, in data memory which is managed by a filing system as clusters or blocks.

The data content objects are beneficially icons representing different functional options, for example in a portable electronic device, but could also be other types of data such as data files, text data, audio data, image data, binary data and measurement data.

Moreover, the present disclosure seeks to provide an electronic device, for example a portable electronic device, which is operable to employ more efficient data storage, for example more efficient data storage of data content objects, in data memory which is managed by a filing system as clusters or blocks.

According to a first aspect, there is provided method of operating a data memory of a device which is managed by a filing system which is operable to store data in respect of one or more clusters or blocks within the data memory, characterized in that the method includes:

(a) assembling together a plurality of data content objects (110, 60) into a virtual container (150);

(b) storing the virtual container (150) and its associated data content objects (110, 60) into one or more of the one or more clusters or blocks (100), wherein the data content objects (110, 60) are memory-aligned within the one or more of

the one or more clusters or blocks (100), wherein cluster or block size is hierarchical such that the one or more clusters or blocks (100) include smaller clusters or blocks with room enough for one or more data content objects smaller than a pre-defined size of a cluster and larger clusters or blocks for data content objects larger than a pre-defined size of a cluster, wherein smaller and larger clusters or blocks are used to do suitable alignment to the data content objects (110,60) based on needs of the used filing system; and

(c) selectively accessing an individual data content object from the plurality of data content objects (110, 60) in the virtual container (150) stored within the data memory (40),

wherein limits of the one or more clusters or blocks (100) are crossed by one data content object (110) of the one or more clusters or blocks if the size of that data content object (110) is larger than a cluster whereat it is stored.

The present invention is of advantage in that there is provided a novel way of storing files into memory in an optimal format, thus enabling near-maximal utilization of a theoretical memory capacity of data storage memory, independently on which particular type of file system is used for managing the data storage memory.

Optionally, the method includes selectively accessing the individual data content object from the plurality of data content objects having mutually different file formats.

Optionally, the method includes transcoding one or more of the plurality of data content objects when storing and/or accessing them from their virtual container stored in the data memory.

Optionally, the method includes assembling together data content objects smaller than a cluster and last blocks of data content objects larger than a cluster within the one or more clusters or blocks.

Optionally, the method includes compressing, encrypting, decompressing or decrypting one or more of the data content objects when storing and/or accessing them from their virtual container stored in the data memory.

Optionally, the method includes arranging for at least one of the plurality of data content objects to include a link to an external database relative to the data memory.

Optionally, in the method, the plurality of data content objects corresponds to data for generating one or more icons for presentation via a graphical user interface (GUI).

According to a second aspect, there is provided a device including a data memory which is managed by a filing system which is operable to store data in respect of one or more clusters or blocks within the data memory, characterized in that the device is operable:

(a) to assemble together a plurality of data content objects (110, 60) into a virtual container (150);

(b) to store the virtual container (150) and its associated data content objects (110, 60) into one or more of the one or more clusters or blocks (100), wherein the data content objects (110, 60) are memory-aligned within the one or more of the one or more clusters or blocks (100), wherein cluster or block size is hierarchical such that the one or more clusters or blocks (100) include smaller clusters or blocks with room enough for one or more data content objects smaller than a pre-defined size of a cluster and larger clusters or blocks for data content objects larger than a pre-defined size of a cluster, wherein smaller and larger clusters or blocks are used to do suitable alignment to the data content objects (110,60) based on needs of the used filing system; and

(c) to access selectively an individual data content object from the plurality of data content objects (110, 60) in the virtual container (150) stored within the data memory (40),

wherein limits of the one or more clusters or blocks (100) are crossed by one data content object (110) of the one or more clusters or blocks if the size of that data content object (110) is larger than a cluster whereat it is stored.

Optionally, the device is operable to access selectively the individual data content object from the plurality of data content objects having mutually different file formats.

Optionally, the device is operable to move together small data content objects and last blocks of larger data content objects within the one or more clusters or blocks.

Optionally, the device is operable to transcode one or more of the plurality of data content objects when storing and/or accessing them from their virtual container stored in the data memory.

Optionally, the device is operable to compress, encrypt, decompress or decrypt one or more of the plurality of data content objects when storing and/or accessing them from their virtual container stored in the data memory.

Optionally, the device is operable to arrange for at least one of the plurality of data content objects to include a link to an external database relative to the data memory.

Optionally, when the device is in operation, the plurality of data content objects corresponds to data for generating one or more icons for presentation via a graphical user interface (GUI) of the device.

Optionally, the device is a portable electronic device.

According to a third aspect, there is provided a computer program product comprising a non-transitory computer-readable storage medium having computer-readable instructions stored thereon, the computer-readable instructions being executable by a computerized device comprising processing hardware to execute a method pursuant to the first aspect.

It will be appreciated that features of embodiments of the disclosure are susceptible to being combined in various combinations without departing from the scope of the invention as defined by the appended claims.

Description of the diagrams

Embodiments of the present disclosure will now be described, by way of example only, with reference to the following diagrams wherein:

FIG. 1 is a schematic illustration of a portable electronic device which is operable to access its data memory pursuant to methods of the present disclosure,

wherein the data memory is organized by its filing system in clusters or blocks;

FIG. 2 is an illustration of data content object storage in clusters of a known data memory;

FIG. 3A is an illustration of data content object storage in virtual containers pursuant to an embodiment of the present disclosure;

FIG. 3B is an illustration of data content object storage that contains several portions of data inside one cluster in virtual containers pursuant to an embodiment of the present disclosure; the limits of the clusters are not crossed by one portion of data unless the size of that particular portion of data is not itself larger than several clusters combined;

FIG. 3C is an illustration of data content object storage where there are several smaller clusters with room enough for one or more portions of data and also larger clusters for larger portions of data;

FIG. 4 is an illustration of implementing additional filing system hierarchy for achieving storage of a plurality of data content objects in an efficient manner in data memory which is organized as clusters or blocks, for example clusters or blocks of fixed pre-determined size.

FIG. 5 is an illustration of two different embodiments pursuant to the disclosure:

(a) a virtual filing system pursuant to the disclosure, implemented by using FUSE, wherein a virtual file system SZIPFS is functioning in user space; and

(b) a virtual file system implemented as an independent filing system, wherein the virtual file system SZIPFS is located in kernel space.

In the accompanying diagrams, an underlined number is employed to represent an item over which the underlined number is positioned or an item to which the underlined number is adjacent. A non-underlined number relates to an item identified by a line linking the non-underlined number to the item. When a number is non-underlined and accompanied by an associated arrow, the non-underlined number is used to identify a general item at which the arrow is pointing.

Description of embodiments of the invention

When describing embodiments of the present disclosure in the following, abbreviations are employed as provided in Table 1:

Table 1: details of acronyms employed to describe embodiments

Acronym	Detail
Block	A physical record, the smallest logical amount of disk space that can be allocated to hold a file.
Cluster	A data cluster is the smallest logical amount of disk space that can be allocated to hold a file.
NFS	Network File System.
NVM	Non-volatile memory: computer memory that can retrieve stored information even after having been power cycled (turned off and back on), e.g. hard disk, magnetic tapes, etc.
RAM	Random Access Memory
ROM	Read Only Memory
Slack space	Wasted disk space.
SSD	Solid State Drive: a solid-state storage device that uses integrated circuit assemblies as memory to store data persistently.
VFS	Virtual File System.
SZIP	Starzip image file.

In overview, embodiments of the present disclosure are suitable, for example, in fields of utilization, namely "*ecosystems*", of the mobile devices and corresponding embedded devices. These ecosystems and their corresponding devices comprise, or depend upon, a considerable amount of files whose size is smaller than a pre-defined fixed-size cluster, namely "*data cluster*", or block, namely "*block as physical record*". The embodiments of the present disclosure also apply to "block devices".

The methods described in respect of embodiments of the present disclosure make it possible to store and include considerably more files than known, namely "*prior art*", systems are capable of accommodating. Moreover, embodiments of the present disclosure make it possible for device manufacturers to produce more competitive, cheaper and more cost-efficient hardware.

Embodiments of the present disclosure are capable of achieving a maximal cost-efficiency in data storage capacity, especially regarding small files that are smaller than a cluster (data cluster) or block (block as physical record) defined in a file system. Thus, the embodiments of the present disclosure are capable of saving considerable amounts of physical memory, namely a performance characteristic that can be important in aforementioned ecosystems and in embedded systems.

Moreover, the present disclosure provides a method of storing data which enables a user to achieve more efficient content compression and encryption and, if necessary, backup copying, for example into another file system or into a centralized data-storage cloud. Therefore, a decoder integrated in a data storage system pursuant to the present disclosure is able to decompress the contents of a file for the user, even if the file were compressed and encrypted. An encoder integrated in the aforementioned data storage system enables transcoding a file from one format into another; namely, a file system in an ecosystem can then request, for example, a *.bmp* image file in the *.png* format, in which case the file is decoded from its original format and re-encoded into the format indicated by a file extension of a link.

Embodiments of the present disclosure can be implemented so that they are compatible with a file system that is most commonly used in ecosystem, thereby not endangering data security in the system and not interfering with its associated user permissions.

Embodiments of the present disclosure differ from known prior art systems in that they are capable of using a file in an optimized fashion directly from an optimized data container that is located in a same physical non-volatile memory, thus not needing to make changes into programs or scripts that use the file; such optimization is advantageously utilized in a portable electronic device including therein data memory, wherein data stored in the data memory is utilized by the device when in operation.

Embodiments of the present disclosure are technically different from known systems of file management for portable electronic devices. For example, in reference [3],

there are produced tailored optimal-sized sequences of data which are incorporated into fixed-sized packets according to their priority and based upon their data format, thereby preventing occurrence of wasted space in the data packets being produced. In contradistinction, embodiments of the present disclosure focus instead on content which has an actual size that is a lot smaller than the size of a cluster (data cluster) or block (block as physical record) used in a given particular file system, wherein such content contemporarily results in slack memory space, namely in inefficient use of memory capacity. Embodiments of the present disclosure are not dependent upon priorities of data, and thus the embodiments of the present disclosure are capable of combining all mutually different kinds of data together. Of course, optionally, embodiments of the present disclosure can still also utilize priorities or assemble similar data contents together as well. Optionally, the embodiments can utilize the knowledge on how often the files are needed, namely how often they are read from memory.

Embodiments of the present disclosure address a technical problem of inefficient utilization of precious memory in such situations, where the "*ecosystems*" provided by device manufacturers are based on known prior art data filing systems and are used so as to maintain compatibility of applications. In these known prior art file systems, the files are stored into one or more pre-formatted fixed-size clusters (data clusters) or blocks (blocks as physical record), thereby enabling a fairly capable and cost-efficient data filing system for reading and writing of medium-sized, and larger than medium-sized files; such known prior art file systems are described, for example, in reference [1]. By "*medium-sized files*" is meant, for example, data files having a size in a range of, for example, from 4 kByte to 1 MByte; by "*larger than medium-sized files*" is meant, for example, data files having a size greater than , for example 1 MByte.

A simple example of such a known prior art file system is as follows: when a file whose size is one byte is stored into a file system where the size of a cluster (data cluster) or block (block as physical record) is by default 4096 bytes, then a hardware non-volatile memory, for example see reference [2], wastes 4095 bytes, namely 99.98%. Such wastage corresponds to gross inefficiency in data storage.

In a given data filing system, the amount of wasted bytes can always be computed when both the size of the file in bytes and the size of a cluster, namely "*a data cluster*", or block, namely "*block as physical record*", used in that particular file system are known. A principal increase in cost-efficiency that embodiments of the present disclosure yield is gained in such file systems where the operating system consists of files smaller than aforementioned medium-sized files, which in known prior art data filing systems operate to cause considerable slack space in hardware non-volatile memories.

For example, the Android operating system uses tens of thousands of images, logos and icons which must be stored in fast non-volatile memories so that they can be loaded fast into Random Access Memory (RAM) of a portable electronic device. Therefore, the majority of such images, logos and icons are smaller in bytes than the pre-defined size of a cluster, namely "*a data cluster*", or block, namely "*block as physical record*", used in that particular file system, because these files are originally small or they have been compressed maximally to fit their purpose in the portable electronic device.

Embodiments of the present disclosure are operable to enable removal of slack space in memory; when the embodiments are employed in known prior art file systems such as:

Ext3, Ext4, NTFS, FAT, NFS, VFS, and similar,

used in several known operating systems:

iOS, Android, OS X, Linux, BSD, Windows, and similar,

a considerable increase in memory usage efficiency is susceptible to being achieved. "*Ext3*", "*Ext4*", "*NTFS*", "*FAT*", "*NFS*", "*VFS*", as well as "*iOS*", "*Android*", "*OS X*", "*Linux*", "*BSD*", and "*Windows*" include registered trademarks.

In the following, several embodiments of the present disclosure will be described. These embodiments are all based on moving all such files that are smaller, originally

or after compression, than the pre-defined size of a cluster (data cluster) or block (block as physical record) into an optimized data container, which however is located physically in the same non-volatile memory as the files. More specifically, at least a last cluster of such files needs to be smaller than a pre-defined size of a cluster, namely "a *data cluster*", or block, namely "*block as physical record*", or at least small enough, as will next be described.

In embodiments of the present disclosure, a small file or a last cluster, namely "*a last data cluster*", or block, namely "*block as physical record*", of a larger file reserves only as many bytes or bits of memory as it needs and thus does not reserve unnecessary slack space from the non-volatile memory. However, it will be appreciated that there is no need to move other than the small files and/or the last clusters of those larger files mentioned above together, so as to enable a fast and easily implementable technical solution. Of course, for example, an entire data content of larger files can also be moved together so that the data content will be in a row. Sometimes, the clusters are moved only when the amount of bytes saved is at least some predefined size; for example, a predefined size of 0.5 kB when 4 kB clusters are used. Such a minimum size rule is employed so as to avoid non-necessary combination of clusters that do not remove enough slack space in data memory. It is also optionally possible to move those last clusters together, so that another, more accurate clustering can be used; for example 0.5 kB clusters are used with 4 kB original clusters. This means that the cluster size is hierarchical, which makes it possible to remove large sections of slack space, although not as much as the maximum savings that can be achieved by using clusters with 1 byte accuracy. Moreover, other cluster sizes can be used, for example smaller and larger, but typically the selected cluster size or cluster sizes should be selected so that they do suitable alignment to the stored data compared to non-aligned 1 byte (8-bit) clusters based on the needs of the used system, for example 32/64/128-bit or 512/1024-byte.

In embodiments of the present disclosure, physical data files can be replaced by links, for example in a manner as described in reference [4], that optionally point to an optimized data container. Moreover, in the embodiments, physical files can be moved to one or more optimized data containers which, as regards the data filing system, are created as virtual files, as a device or as parallel data filing systems.

The clusters, namely "*data clusters*", or blocks, namely "*blocks as physical record*", reserved by files moved in this way are set free in the data filing system and they are technically copied into clusters, namely "*data clusters*", or blocks, namely "*blocks as physical record*", reserved by an optimized data container in the same non-volatile memory. Embodiments of the present disclosure are optionally also used for volatile memories, when they are also read using a filing system access. Typically those volatile memories are not read using a filing system access.

Embodiments of the present disclosure make it possible to achieve a cost-efficient technical solution in many different operating systems, for example of portable electronic devices as aforementioned, thereby paying attention to the user permissions and other important basic functionalities of a data filing system. It will be appreciated that the data filing system pursuant to the disclosure, also known as a "file system", can also function as an independent filing system just like the contemporary filing systems, and is susceptible to being implemented at a hardware level or as a software driver, but resulting in even better overall data storage efficiency and enhanced data storage capacity for given memory.

On the other hand, embodiments of the present disclosure are concerned with a software solution which enables integration into an existing data filing system software that can be executed both in an administrative mode, namely "*admin mode*", and in a user mode; with regard the user mode, reference [5] is concerned with "*user space*". If embodiments of the present disclosure are integrated into a kernel of an operating system, then it will become a part of the operating system, namely being executed with permissions of the operating system, as part of its associated kernel space. It is also possible to implement embodiments of the present disclosure in the kernel space that is run in the kernel of the operating system, if the files whose space usage is optimized need elevated permissions or are physically located in another memory, for example in a section of data memory reserved for firmware.

In certain operating systems, embodiments of the present disclosure can be implemented also as a background service, but in such a case it is a software application being executed in the operating system that is responsible for running the implementation and that is mounted, see reference [6], in the data filing system,

namely "*file system*", as a virtual drive or as a virtual device; see reference [7] with regard to virtual file systems.

If it not possible to integrate embodiments of the present disclosure into a file system of a given operating system, either because of strict operating conditions set by an associated device manufacturer or because it would not otherwise be possible to implement technically, then the embodiments can still be implemented by using a network drive; network drives are described in reference [8] in respect of network file systems. However, a NFS-based system must be implemented in such a way that contents of a given optimized data container reserves drive space in the same file system.

Optionally, embodiments of the present disclosure can be implemented so that only such data files that have mutually similar permissions are moved together. Optionally, information about the file permissions are moved together with the files themselves, which makes it easier to restrict unauthorized usage of the data files, as associated administrator's desire.

When contemporary operating systems are principally based on *NIX systems, embodiments of the present disclosure are susceptible to being used for the Linux operating system, since such implementation can be adapted with minor changes, also on Android and iOS mobile platforms for example. "**NIX*", "*Android*" and "*iOS*" are registered trademarks.

Regardless of which technical implementation is used, the files to be optimized need to be assembled into the data container. The data container can be a simple file or a database, such that embodiments of the present disclosure are beneficially used when:

(i) data files are smaller than a cluster (data cluster) or block (block as physical record), wherein the data files are stored into the data container; and

(ii) the last cluster or block of files larger than a cluster is stored into the data container; or

(iii) the entire file is stored into the data container, which is located physically on a data storage drive/disk as the optimized files.

A very simple solution pursuant to the present disclosure is a virtual file that has the size of one file which simulates compatibility with the existing file system. However, this virtual file in itself comprises an optimal way to store files without the current prior art limitation of having a minimum size for a cluster (data cluster) or block (block as physical record). Such a functionality can be implemented, for example, by using a proprietary FUSE program, described in reference [11], which offers a simple interface for user space applications for exporting a virtual file system to the Linux kernel. An example of a virtual filing system pursuant to the disclosure, implemented by using FUSE (https://en.wikipedia.org/wiki/Filesystem_in_Userspace) is illustrated in FIG. 5a, wherein a virtual file system SZIPFS is operable to function in a user space, so that FUSE executes software code that needs elevated kernel space privileges. A command used is 'ls -l /home/jed', which is used to give a detailed list of files in a directory '/home/jed' maps to the virtual filing system, using a starzipfs computer program that employs embodiments of the present disclosure and thus lists the files there as if they were in '/home/jed'.

Next, there will be provided an explanation on how such a virtual file system is implemented as an independent filing system comparable to, for example, Extent and NFS (see reference [8]). This scenario is illustrated in FIG 5b, wherein SZIPFS is located in a kernel space. In such a scenario, mounting SZIPFS needs elevated privileges. SZIPFS employs a filing system that needs to be created using admin permissions, but a user is able thereafter to use corresponding SZIPFS disk space with methods pursuant to the present disclosure, as if it were, for example, an EXT4 journaling file system (see https://en.wikipedia.org/wiki/Ext4).

In an example embodiment of the present disclosure, the initialization, mounting and usage of a virtual file system is simple and easy technically, by giving commands in two phases. It will be appreciated that such a procedure can also be automated and configured in an '/etc/fstab' file, for example as described in reference [12]. Following steps are thus executed:

(i) creating a mount point directory for an optimal data container, for example by giving a command: *mkdir /mnt/szip*; and

(ii) linking the optimal data container to the directory being used, for example by giving a command : *./starzipfs /home/jed /mnt/szip*.

In such an implementation, applications and/or libraries have the option to directly overwrite standard input/output operations (I/O) of file handling, such as the C functions:

fopen, fseek, fread, fwrite, fclose

but not limited thereto. This option enables pointing/linking to a file inside an optimized data container, instead of pointing/linking directly to a file. The described implementation is optimal if the optimized data container is used only for a certain solution or application, and there is therefore no need to make changes into the operating system such as to modify user permissions, or to install and/or start software programs necessary for implementing the service, something that needs to be done for example when implementing a data filing system using aforementioned FUSE.

It is also optionally possible, pursuant to the present disclosure, to overwrite the standard I/O functions of file handling: :

(a) in advanced programming language software code, in which case the changes affect only the application and/or library internally; or

(b) in the application Programming Interface (API) of the operating system, in which case the changes may also affect external applications and/or libraries.

It will be appreciated that, with embodiments pursuant to the present disclosure, there is no need to overwrite the I/O functions of file handling if files of the optimized data container are used, in which case a programmatic interface provided by the optimized data container can be used directly.

In a recent 2015-08 version of the Android operating system (OS), namely open source software as described in reference [9], there are included 22834 images, namely icons, logos and other graphics that are installed into the file system of devices along with the OS; "*Android*" is a registered trademark. These images are

used in the construction of a principal user interface, namely a principal GUI. There will next follow two examples in which it is assumed that the image files were stored into a physical solid state drive (SSD) memory whose block size is by default 4096 bytes; with regard to SSD, see reference [10]. Moreover, in the examples, the files are stored as PNG images.

In a first example pursuant to the present disclosure, all such images are selected whose size in bytes, namely their actual size, is either of a mutually same size as that of the cluster (data cluster) or block (block as physical record), or smaller than that, namely all together 17846 PNG files. The content of these files all together is 19982717 actual bytes, but they reserve 73097216 bytes on disk, which means that using embodiments of the present disclosure releases a total of 53114499 bytes of slack space to be used more productively, namely 72.66%, which is a considerable improvement as regards the usage of physical memory.

In the second example pursuant to the present disclosure, all aforementioned images of the first example are selected, irrespective of their content size, namely all together 22834 PNG files. The content of these files all together is 283565760 actual bytes, but they reserve 347996160 bytes on disk, which means that using embodiments of the present disclosure is capable of releasing a total of 64430400 bytes to be used more productively, namely 18.51%.

In both the aforesaid first and second examples, it is possible to achieve a considerable improvement in data memory utilization.

It will be appreciated that if these files were to be compressed with, for example, a proprietary GMVC® codec, produced by Gurulogic Microsystems Oy, Finland, the GMVC® codec typically compressing approximately two times better that a contemporary PNG codec, then embodiments of the disclosure are capable of removing even more slack data file space, and thus even more disk space is saved for more productive use. Moreover, it will further be appreciated that even though these simplest examples receive a file in the PNG format and store it on disk in re-organized manner also in the PNG format, it is also possible to execute data format conversion as well in connection with the reading and the writing of the file, on

condition that such a conversion can be performed fast enough so that it does not interfere with the user experience. Thus, for example, a PNG file can, in a solution pursuant to the present disclosure, first be converted into a .gmvc file which is then stored on disk using methods pursuant to the present disclosure. Then, as the user or the system desires to inspect the file, it can be read out of the system as a PNG file, which means that the inverse conversion from GMVC to PNG format needs to be executed in connection with reading the file. Of course, the user or the system may potentially wish to read the file content in the .gmvc format, in which case the format conversion is not necessary in an associated data reading process.

Optionally, it is also possible that the user or the system is desirous to render an image on a device display from the file system, and in that case it is advantageous to execute associated transcoding directly from the *.gmvc* file into a BGRA or RGBA format, namely without a need firstly to convert the file to .png format and then to BGRA or RGBA format, which are directly usable at the input of the display device; such a situation is usually not the case with, for example, PNG or GMVC® formats.

Referring to FIG. 1, there is shown an illustration of a portable electronic device, for example a smart phone, a gaming terminal, a personal instrumentation device, a medical diagnostics device or similar, indicated generally by **10**. The portable electronic device **10** includes a data processing arrangement **20** including a data processor **30** and its associated data memory **40**. Moreover, the data processing arrangement **20** is coupled to a graphic display arrangement **50** which is operable to provide a graphical user interface (GUI). Optionally, the GUI is implemented as a touch-screen.

In operation, data is stored in, and accessed from, the data memory **40**, wherein the data processing arrangement **20** employs a data filing system, also known as a "*file system*", which organises the data memory **40** into clusters or blocks, as aforementioned. The data processing arrangement **20** is operable to execute one or more software applications, known as "Apps", for enabling the data processor arrangement **20** to perform one or more user-defined functions. In operation, one or more graphical symbols **60**, known as "icons", are shown on the graphical user interface (GUI); data required for generating the one or more graphical symbols **60**

are stored in the data memory **40**, wherein data for a given graphic symbol **60** is stored in a corresponding given cluster of the data memory **40**. The user is able to invoke the one or more software applications (Apps) by touching corresponding one or more graphical symbols **60** of the GUI, whereafter the one or more invoked software applications are executed on the data processor **30**.

Referring next to FIG. 2, there is shown an illustration of a portion of the data memory **40** which is organized in a conventional manner by the aforementioned data filing system into clusters, denoted by **100A**, **100B**, **100C** and so forth, wherein the clusters **100A**, **100B**, **100C** are operable to have stored therein corresponding data **110A**, **110B**, **110C** corresponding to the one or more graphic symbols **60**, with corresponding slack spaces **120A**, **120B**, **120C**. When the data **110A**, **110B**, **110C** is considerably smaller in size to that of the clusters **100A**, **100B**, **100C** respectively, the data memory **40** is utilized very inefficiently. Moreover, when data compression is employed to compress the data **110A**, **110B**, **110C** within their respective clusters **100A**, **100B**, **100C**, utilization of the data memory **40** becomes even more inefficient. This means that when compression is applied to data, it does not offer any benefit to the used disk space, when the amount of data is originally below the size of a cluster.

07 02 18

Referring next to FIG. 3A, the data **110A**, **110B**, **110C** are concatenated within a virtual container **150** which is then stored in one or more of the clusters **100A**, **100B**, **100C** with relatively little slack data space in the data memory **40**, representing a considerable improvement in utilization of the data memory **40**, despite the data memory **40** and its associated filing system continuing to manage the data memory **40** in terms of clusters **100**. An advantage of such an approach is that applying compression to the data **110** in the virtual container **150** is capable of further improving utilization of the data memory **40**, in contradistinction to known filing systems employing clusters not benefitting from such compression, as aforementioned. Despite the data **110** being concatenated into the clusters **100** as illustrated in FIG. 3, embodiments of the present disclosure provide for the data **110** within the container to be searchable and extractable, for example for generating one or more of the graphical symbols **60**.

Referring next to FIG. 3B, embodiments of the present disclosure are operable to create an environment in which several portions of data **110A'**, **110B'**, **110C'** are stored inside one cluster in virtual containers pursuant to an embodiment of the present disclosure. The limits of the clusters are not crossed by one portion of data unless the size of that particular portion of data is not itself larger than several clusters combined, as is indeed the case with the portion of data **110D'**.

Referring next to FIG. 3C, embodiments of the present disclosure are operable to create an environment in which there are several smaller clusters with room enough for one or more portions of data **110A''**, **110B''**, **110C'** and also larger clusters for larger portions of data **110D''**. This kind of solution is well suited for cases where a memory-aligned and hierarchical clustering scheme is desired.

Additionally, optionally, at least a portion of the data **110**, or links to such, are copied also to an external device, in order to enable using the data in other devices, for example in a variety of devices of the same user, for example when transferring data from a personal computer to a smart phone, or from a smart phone to a wearable digital electronic device such as a smart watch.

It will be appreciated that even if the embodiments pursuant to the disclosure are capable of using a file in an optimized fashion directly from an optimized data container that is located in a physical non-volatile memory of the same device, it is possible that also initially storing the files, such as icons representing different functions, to the device can take place directly in an optimized manner, instead of storing the data in a known way and thereafter optimizing the storage of the data in the device.

Referring next to Table 2, there is shown an example of file name, with corresponding file size in disk data memory, with corresponding file size in bytes.

Table 2: Linking example

File name	File size in disk	File size in bytes

border.png	4096	67
alpha.9.png	4096	117
white.png	4096	842
holo6.png	4096	3070
…	…	…

In Table 3, there is shown an example of file name, file size in disk data memory, and corresponding file size in bytes.

Table 3: Linking example

File name	File size in disk	File size in bytes
/dev/szip	4096	4096
border.png	0	67
alpha.9.png	0	117
white.png	0	842
holo6.png	0	3070
…	…	…

As illustrated above in Tables 2 and 3, the files on the left-hand-side are moved to an optimal data container **150** and replaced with a symbolic link, so that they can still be found in the file system. It will be appreciated that the physical actual file size of the files in the left-hand-side table is only 4096 bytes, because the contents of the files are linked to a virtual file called /dev/szip.

It will be ~~be~~ appreciated that this kind of “virtual file names” such as above make it possible firstly to open a file and, after the opening, to read/use the file. That is, the embodiments pursuant to the present disclosure do not need to know the actual file name anymore, yet the file in question still physically exists. Further aims of using such a virtual file name are not always needing to store the entire file name. Instead, searching for the file is optionally executed based upon some sort of virtual identification or modified file name, whereby the actual file name is always acquired, for example, from the file system when necessary. Such an identification or file

handle is optionally stored, for example, into a database, so as to enable identification and processing of the file.

Now, from a point of view of regular filing systems, the physical file exists in a virtual file system., pursuant to the present disclosure In principle, all the files of the embodiments pursuant to the disclosure are virtual from a point of view of the regular file system, even though the actual names were known. Indeed, the file names from a point of view of the regular filing systems are always actual names,

Referring next to Table 4, there is provided further information regarding content inside /dev/szip and a corresponding offset employed in embodiments of the present disclosure; Table 4 thus provides, for example, information regarding an image file.

Table 4: Image file details

Content inside /dev/szip	Offset
meta-data to files(s) offsets	0 (total 47 bytes)
border.png()	47
alpha.9.png	114
white.png	231
holo6.png	1073
……	…

Table 4 presents the image file /dev/szip which stores the files into one physical data container.

Referring next to FIG. 4, the container **150** is generated by operation of digital hardware components and/or by operation of one or more software layers which cooperate with the file system, and is optionally an integral part of the file system. For example, in an embodiment of the present disclosure wherein the file system is implemented as a software layer **200**, the file system is operable to store data in clusters **100**. There is utilized an additional software layer **210** for implementing the container **150** for achieving more efficient data storage in the data memory **40**, pursuant to a regime as illustrated in FIG. 3A, FIG, 3B and FIG. 3C. The additional software layer **210** is operable to assist with accessing the individual data **110**, as

well as ensuring that it is concatenated, or otherwise stored compactly, within the clusters **100**.

Thus, embodiments of the present disclosure are susceptible of achieving maximal cost-efficiency in data storing capacity, especially with regard to small files that are smaller than the cluster (data cluster) or block (block as physical record) defined in a file system **200**. Considerable amounts of physical memory are saved during data storage, namely which can be important in ecosystems and in embedded systems.

Moreover, the file storing method described in the present disclosure enables for the user efficient data content compression and encryption and, if necessary, backup copying, for example into another file system or in a centralized cloud, as aforementioned. Therefore, a decoder integrated in the data storing system is able to decompress the contents of a file for the user even if the file is compressed and encrypted. An encoder integrated in the data storing system enables transcoding a file from one format into another; namely a file system in an ecosystem can then request, for example, a *.bmp* image file in the *.png* format, in which case the file is decoded from its original format and re-encoded into the format indicated by the file extension of the link.

Embodiments of the present disclosure can be implemented so that they are compatible with the file system most commonly used in a given "*ecosystem*", thereby not endangering data security in the system and not interfering with user permissions.

The method pursuant to the disclosure is optionally implemented in two ways, as regards memory allocation. One way is to reserve/allocate a disk for purposes of a file storing system pursuant to the present disclosure, in a similar manner as, for example, TrueCrypt or the Stacker disk compression utility function. Another beneficial way is to implement the method in such a way that there is no need initially to reserve a disk, but instead files that do not use the method pursuant to the disclosure can co-exist with files that use it, and thus the method pursuant to the disclosure utilizes clusters one by one when needed, namely in a selectively invoked manner. Therefore, it is easy to move even existing data to be used on a disk

optimized with this method, one file at the time. This can be executed even in a background process, without losing disk space at any point by pre-allocating/pre-reserving existing disk space. Moreover, since the method pursuant to the disclosure can use existing disk space, it addresses a problem that known data filing systems often have, namely a problem of not being able to perform the pre-reservation of space because the disk is already full.

In the foregoing, the device **10** is operable to allow addressing the data content objects **110**, **60** via use of virtual files, for example.

Modifications to embodiments of the invention described in the foregoing are possible without departing from the scope of the invention as defined by the accompanying claims. Expressions such as “including”, “comprising”, “incorporating”, “consisting of”, “have”, “is” used to describe and claim the present invention are intended to be construed in a non-exclusive manner, namely allowing for items, components or elements not explicitly described also to be present. Reference to the singular is also to be construed to relate to the plural. Numerals included within parentheses in the accompanying claims are intended to assist understanding of the claims and should not be construed in any way to limit subject matter claimed by these claims.

REFERENCES

[1] Data cluster - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Data_cluster

[2] Non-volatile memory - Wikipedia, the free encyclopedia (accessed August 17, 2015). URL: https://en.wikipedia.org/wiki/Non-volatile_memory

[3] Method of Communication Data Packets Within Data Communication Systems (March 2015). Patent Application, GB 1504336.7.

[4] link (Unix) - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Link_%28Unix%29

[5] User space - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/User_space

[6] Mount (computing) - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Mount_%28computing%29Network File System - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Network_File_System

[7] Virtual file system - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Virtual_file_system

[8] Network File System - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Network_File_System

[9] Building the System | Android Open Source Project (accessed August 17, 2015): URL: https://source.android.com/source/building.html

[10] Solid-state drive - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Solid-state_drive

[11] FUSE: Filesystem in Userspace (accessed August 17, 2015): URL: http://fuse.sourceforge.net/
https://en.wikipedia.org/wiki/Filesystem_in_Userspace

[12] fstab - Wikipedia, the free encyclopedia (accessed August 17, 2015): URL: https://en.wikipedia.org/wiki/Fstab

CLAIMS

1. A method of operating a data memory (40) of a device which is managed by a filing system (200) which is operable to store data (110) in respect of one or more clusters or blocks (100) within the data memory (40), characterized in that the method includes:

(a) assembling together a plurality of data content objects (110, 60) into a virtual container (150);

(b) storing the virtual container (150) and its associated data content objects (110, 60) into one or more of the one or more clusters or blocks (100), wherein the data content objects (110, 60) are memory-aligned within the one or more of the one or more clusters or blocks (100), wherein cluster or block size is hierarchical such that the one or more clusters or blocks (100) include smaller clusters or blocks with room enough for one or more data content objects smaller than a pre-defined size of a cluster and larger clusters or blocks for data content objects larger than a pre-defined size of a cluster, wherein smaller and larger clusters or blocks are used to do suitable alignment to the data content objects (110,60) based on needs of the used filing system; and

(c) selectively accessing an individual data content object from the plurality of data content objects (110, 60) in the virtual container (150) stored within the data memory (40),

wherein limits of the one or more clusters or blocks (100) are crossed by one data content object (110) of the one or more clusters or blocks if the size of that data content object (110) is larger than a cluster whereat it is stored.

2. A method as claimed in claim 1, characterized in that the method includes selectively accessing the individual data content object from the plurality of data content objects (110, 60) having mutually different file formats.

3. A method as claimed in claim 1 or 2, characterized in that the method includes assembling together data content objects (110) smaller than a cluster and last blocks of data content objects (110) larger than a cluster within the one or more clusters or blocks (100).

4. A method as claimed in any one of the preceding claims, characterized in that the method includes transcoding one or more of the plurality of data content objects (110, 60) when storing and/or accessing them from their virtual container (150) stored in the data memory (40).

5. A method as claimed in any one of the preceding claims, characterized in that the method includes compressing, encrypting, decompressing or decrypting one or more of the data content objects (110, 60) when storing and/or accessing them from their virtual container (150) stored in the data memory (40).

6. A method as claimed in any one of the preceding claims, characterized in that the method includes arranging for at least one of the plurality of data content objects (110, 60) to include a link to an external database relative to the data memory (40).

7. A method as claimed in any one of the preceding claims, characterized in that the plurality of data content objects (100, 60) corresponds to data for generating one or more icons for presentation via a graphical user interface (GUI, 20).

8. A device (10) including a data memory (40) which is managed by a filing system (200) which is operable to store data (110) in respect of one or more clusters or blocks (100) within the data memory (40), characterized in that the device (10) is operable:

(a) to assemble together a plurality of data content objects (110, 60) into a virtual container (150);

(b) to store the virtual container (150) and its associated data content objects (110, 60) into one or more of the one or more clusters or blocks (100), wherein the data content objects (110, 60) memory-aligned within the one or more of the one or more clusters or blocks (100), wherein cluster or block size is hierarchical such that the one or more clusters or blocks (100) include smaller clusters or blocks with room enough for one or more data content objects smaller than a pre-defined size of a cluster and larger clusters or blocks for data content objects larger than a pre-defined size of a cluster, wherein smaller and larger clusters or blocks are used to do suitable alignment to the data content objects (110,60) based on needs of the used filing system; and

(c) to access selectively an individual data content object from the plurality of data content objects (110, 60) in the virtual container (150) stored within the data memory (40),

wherein limits of the one or more clusters or blocks (100) are crossed by one data content object (110) of the one or more clusters or blocks if the size of that data content object (110) is larger than a cluster whereat it is stored.

9. A device (10) as claimed in claim 8, characterized in that the device (10) is operable selectively the individual data content object from the plurality of data content objects (110, 60) having mutually different file formats.

10. A device (10) as claimed in claim 8 or 9, characterized in that the device is operable to move together small data content objects (110) and last blocks of larger data content objects (110) within the one or more clusters or blocks (100).

11. A device (10) as claimed in any one of claims 8 to 10, characterized in that the device (10) is operable to transcode one or more of the plurality of data content objects (110, 60) when storing and/or accessing them from their virtual container (150) stored in the data memory (40).

12. A device (10) as claimed in any one of claims 10 or 11, characterized in that the device (10) is operable to compress, encrypt, decompress or decrypt one or more of the plurality of data content objects (110, 60) when storing and/or accessing them from their virtual container (150) stored in the data memory (40).

13. A device (10) as claimed in any one of claims 8 to 12, characterized in that the device (10) is operable to arrange for at least one of the plurality of data content objects (110, 60) to include a link to an external database relative to the data memory (40).

14. A device (10) as claimed in any one of claims 8 to 13, characterized in that the plurality of data content objects (100, 60) corresponds to data for generating one or more icons for presentation via a graphical user interface (GUI, 20) of the device (10).

15. A device (10) as claimed in any one of claims 8 to 14, characterized in that the device (10) is a portable electronic device.

16. A computer program products comprising a non-transitory computer-readable storage medium having computer-readable instructions stored thereon, the computer-readable instructions being executable by a computerized device comprising processing hardware to execute a method as claimed in any one of claims 1 to 7.