IMPROVING I/O PERFORMANCE IN WEB BROWSER ENGINE OF SMART TV PLATFORM

Cheolhee Lee, Youjip Won

Department of Computer Software Hanyang University, Seoul, Korea
lch6719@hanyang.ac.kr, youjip.won@gmail.com

Abstract: Smart TV applications use a web browser to execute xml, html, css, javascripts, images, and videos, which are stored in NAND flash, and load them into memory. The applications also convert resources into a tree data structure. This converted data structure is placed in the heap area and is removed from memory when the application is ended. Therefore, when the application is restarted, this converting process is performed again which slows the speed of a web browser. This paper suggests a technique called fast IO to add persistency to the data tree so that it can be saved and reused when the application is restarted. With fast IO, a persistency area is called an object. It is created the object file and the data file is saved and reused. Storing a tree-data structure in an object adds persistency to the data structure which means it can be reused when the application is restarted after being ended. Applying fast IO in the web browser reduced the time it took to open web pages by up to 68% by omitting the processing of re-forming the tree data structure.

Keywords: smart TV application, web browser engine, DOM Tree, memory mapped file, persistency object

1 Introduction

A smart TV is composed of CPU, DRAM, and NAND flash. Web based smart TVs execute applications that are loaded by a web browser engine [1], e.g., video player, web browser, games, etc. In web browser engines, html files are parsed to build a tree [2] (Fig. 1). A web browser engine displays a web page on smart TV's screen and the nodes of a tree contain information on the web page's location, size, and colors of each elements [3]. When the application is terminated, the tree is removed from memory. When the application is restarted, resources stored in NAND flash need to be converted to a tree form again.

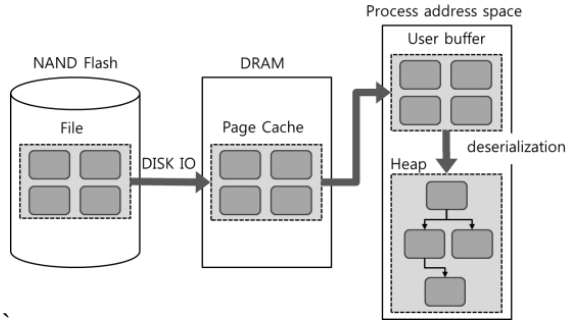

Figure 1 Deserialization process in legacy web browser

978-1-4799-4734-8 /14/$31.00 ©2014 IEEE

This study suggests a technique of adding persistency to selective memory data dynamically allocated from a smart TV. Persistency in this study means that data in process address space can be conserved and reused even after the corresponding application is eliminated. To add persistency to memory data, an object is presented in persistent area. An object exists in process address space and is mapped to a file through mmap () system call. The mapped file is called an object file. By mapping object files to process address space through mmap () system call, the objects can be reused when the application is restarted.

Every time a process is executed, the object file is mapped to a random location in the process address space. When the data structure is saved in a file, pointer validation between nodes composing the data structure of process address space is lost. To ensure that pointer validation is not lost, persistency object file is mapped to fixed process address space. An object provides allocation/removal interface in byte unit. It is implemented based on malloc () algorithm of glibc. To reuse an object, naming system is implemented for its memory data structure. Each object has a name and the names are managed through naming system. In order to store naming system information, metadata object file is created and used.

2 Related work

In ordinary systems composed of CPU, DRAM, and block device, data are conserved by saving them in files and when the data need to be reused, the files are read by memory. This process is conducted through file system. Ext2 [8] is the file system of Linux. FAT [9] was designed for floppy disks and is a file system used in DOS or Windows 9x. Persistency can be added in data through database in which data existing in memory is saved in the disk and is used by read by memory when needed. Data in disk and data structure of memory are different. Therefore, method adding persistency in data using file system and data base requires deserialization. SoftPM [10] automatically adds persistency to data structure connected with the root node. NVRAM can be used as storage, replacing existing HDDs and SSDs, and it can also be used as main memory instead of DRAM. Mnemosyne [11] and NV-Heaps [12] are studies on adding persistency to data structure. These provide process address space

that has persistency and selectively add persistency to data to be reused without deserialization process.

3 Design

This chapter demonstrates how to provide persistency to the tree data structure and how to reuse it when a web browser application is restarted after being ended. In order to add persistency to a memory object, a file is created to store the memory object and this file is saved in the process address space by mmap () system call. This creates a persistent storage space which is called an object. An object is mapped to a file called an object file (Fig. 2). In addition, an interface that allocates memory, in bytes, within the object is created.

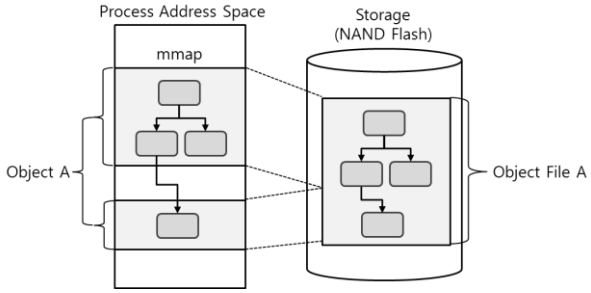

Figure 2 Object and object file

3.1 Pointer validation problem

To reuse an object, its object file should be placed in the address space of the process using mmap () system call. However, when the application is ended, the value of the pointer address showing the connection between the nodes becomes invalid. There are two ways to solve this problem. The first method is to use a relative address without fixing the address of an object, instead of using an absolute address. The second method is always using the same address space to map an object. The first method requires swizzling [4] process because an object does not use an absolute address. Since this incurs additional overhead, this study uses the second method to ensure pointer validation.

3.2 The shared library and collisions between objects

When an application is executed, the address space of the application is generated. At this time, shared library used by the application, such as libc, is loaded into the mmap area. On Linux, shared library is assigned to an address called the mmap_base. mmap_base is the starting address of mmap area and is determined by adding a random value to 0xB7701000. When address space layout randomization [5, 13] is applied, mmap_base changes with every run of a process and the address to which shared library is loaded also changes. As the location of shared library changes, one needs to calculate where the library is loaded and assign any objects created in this area to a fixed address to solve any conflicts between shared library and the objects. To prevent objects from being created in the same space as shared library, addresses around shared library is reserved.

3.3 Namespace for re-mapping memory object

An object file is accessible via the file system's namespace. However, it is impossible to know the location of the object to which the file is mapped. Thus, a namespace to manage objects is provided and it remaps objects to object files, using objects' names. The address of an object can be identified by the object's name and the object can be accessed via this address. Metadata object file, which maintains and stores the name space information, is generated in the file system.

3.4 Object reservation

An object places its object file in the address space of the process in mmap area using the mmap () system call. Each object, to ensure its validity, should always be positioned in the same address space. However, if the address space is already in use for other usage, it is impossible to place the object in its address space. In order to prevent this, the object's address space needs to be reserved. Therefore, when the program is started, all namespaces are visited and address spaces that are used by an object are reserved so that they are not used for other purposes.

3.5 Node allocation

Allocating nodes in an object is based on memory allocation algorithm of glibc. There is metadata in the beginning region of each object and it manages the free chunk by size. When memory allocation is requested, in bytes, it searches the namespace, finds metadata of the object, checks the free chunk in the object, and allocates memory according to the requested size. If this request is not satisfied because of insufficient empty space, system allocates new region, ramps up the object, and allocates memory. In a heap, increases in address space are continuous. However, for an object, increases may be discontinuous because continued address space may already be in use.

4 Evaluation

This chapter explains performance evaluation of fast IO in smart TVs. The experiment measured the time it took to execute a simple web browser, Dillo2.2 [6], before and after applying fast IO technique. The test environment was AMD Phenom X4 925 Processor and DDR3 DRAM 4GB. For storage, an SSD (60 GB OCZ VERTEX2 SATA2), RAM disk, and 500 GB, 720 RPM hard disk were used.

Research on web browsing pattern shows that web users frequently visit the same site [7]. This makes cache a great influence on the performance of a web browser. With fast IO method, the cache mechanism of the web browser was modified. In the past, the web browser (Legacy Dillo) downloaded html and images from the web and saved them in the disk cache. Then, a tree was composed with html. The web page was displayed on the screen via the tree and the tree data structure disappeared when the web browser was ended. When web pages were re-accessed, resources saved in the disk were loaded into memory and the processes described above were performed again. fast IO web browser (F-Dillo) in smart

TVs acquires an html when accessing a web page. It applies fast IO technology and caches the data tree structure. Therefore, when the web browser is restarted and the web page is re-accessed, it is possible to omit the tree building process by reusing the data structure tree. The performance was measured by the time from starting a web browser to displaying the web pages on the screen.

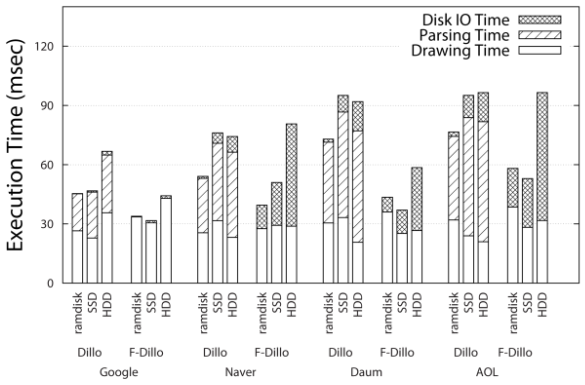

Figure 3 Execution time

Fig. 3 shows disk IO time, parsing time, and drawing time for opening four different web pages. For each page, two methods were used to open the website: an existing web browser, marked as Dillo, and a web browser with fast IO, marked as F-Dillo. The experiment was also performed using a variety of storage devices such as RAM disk, an SSD, and an HDD. Fig. 3 is the case using a hard disk as storage. Using fast IO technique removed parsing time for all web pages. However, disk IO time increased significantly when using a hard disk as storage. This is due to the overhead caused by saving a tree data structure which can be 10 times larger than the file size of an html. Since hard disks have slow IO speed, the increased overhead in disk IO is greater than the effect of omitting parsing process. From this result, it can be confirmed that fast IO approach is not appropriate when using a hard disk for storage. When the experiment used an SSD as storage, it omitted the parsing time and increased disk IO time, as with an HDD. However, because SDDs have fast IO speed, performance improvement by omitting parsing time is larger than the overhead increase in disk IO. Using RAM disk also eliminated parsing time while reducing overhead increase in disk IO compared to when using an SSD. The overall performance improvement by using fast IO technique was most significant in the SSD environment with running time reduction of up to 68%.

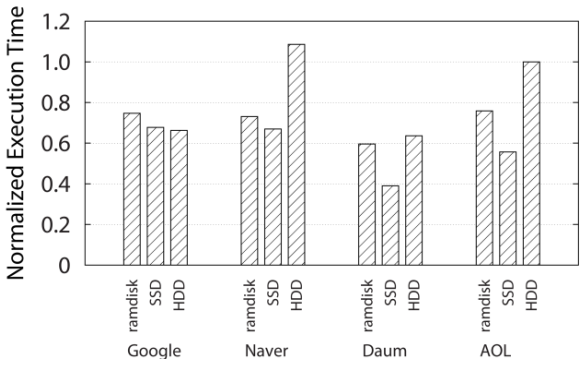

Figure 4 Normalized execution time

Fig. 4 shows execution time of F-Dillo, normalized against that of the legacy Dillo. In case of the HDD, some web pages showed degraded performance due to the increased overhead in disk IO. On the other hand, in an SSD environment, all web pages showed performance enhancement with F-Dillo with the maximum of 61% improvement than HDD.

5 Conclusions

This paper proposes using a fast IO technique in smart TV environment. It is possible to impart persistency to selected memory data that have been allocated dynamically by using the techniques of fast IO. With fast IO technique, smart TV application caches the tree form data and reuses it when the application is restarted. This eliminates the process of converting data into the tree format when the application is restarted. However, as shown in the experimental results, storing tree data structures increases the overhead in disk IO. Therefore, applying fast IO technique is not appropriate when using storage with slow input/output speed, such as hard disk drives, because performance decrease from increased overhead in disk IO more than offsets performance improvement gained by reusing tree form data. On the other hand, when using storage with fast input/output features, such as SSDs, overhead increase in disk IO becomes relatively small and applying fast IO technology will result in overall performance gain. Since smart TVs use NAND flash memory, which guarantees high input/output speed, applying fast IO technology is expected to bring performance improvement.

Acknowledgements

New Memory: This work was supported by IT R&D program MKE/KEIT (No. 10041608, Embedded System Software for New-memory based Smart Device).

References

[1] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. “Web Browser as an Application Platform: The Lively Kernel Experience.” Technical Report. Sun Microsystems, Inc., Mountain View, CA, USA. 2008.

[2] T. Lam, J. Ding, and J.C. Liu. Xml document parsing: Operational and performance characteristics. IEEE Computer, 41(9):30–37, Sept. 2008.

[3] K. Zhang, L. Wang, A. Pan, and B.B. Zhu. Smart caching for web browsers. In Proc. of the 19th international conference on World wide web, Raleigh, NC, USA, Apr. 2010. 24

[4] J. Eliot, B. Moss. Working with persistent objects: To swizzle or not to swizzle. Software Engineering, IEEE Transactions on, 18(8):657–673, Aug. 1992

[5] H. Shacham, M. Page, B. Pfaff, E.J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of addressspace randomization. In Proc. of the 11th ACM conference on Computer and communications security, Washington. DC. USA, Oct. 2004.

[6] J. Arellano-Cid and Von Brand H.H. Network programming internals of the dillo web browser. In Computer Science Society, 2000. SCCC’00. Proceedings. XX International Conference of the Chilean, Santiago, Nov. 2000.

[7] E. Adar, J. Teevan, and S.T. Dumais. Resonance on the web: web dynamics and revisitation patterns. In Proc. of the 27th international conference on Human factors in computing systems, Boston, MA, USA, Apr. 2009.

[8] Card, Remy, Theodore Ts'o, and Stephen Tweedie. "Design and implementation of the second extended filesystem." Proceedings of the first Dutch international symposium on Linux. 1994.

[9] Mitchell, Stan. Inside the Windows 95 file system. O'Reilly & Associates, Inc., 1997

[10] J. Guerra, L. M´armol, D. Campello, C. Crespo, R. Rangaswami, and J. Wei. Software persistent memory. In Proc. of the USENIX ATC '12, Boston, MA, June. 2012.

[11] H. Volos and M. Swift. Mnemosyne: Lightweight persistent memory. In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Newport, California, USA, Mar. 2011.

[12] J. Coburn, A. Caulfield, A. Akel, L. Grupp, R. Gupta, R. Jhala, and S. Swanson. Nv-heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In Proc. of the ASPLOS , Newport, California, USA, Mar. 2011.

[13] V. Lvin, G. Novark, E. Berger, and B. Zorn. Archipelago: trading address space for reliability and security. In In Proc. of the ASPLOS'08, Seattle, Washington, USA, Mar. 2008.