International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 3, Number 5 (2013), pp. 405-410

http://www. irphouse.com /ijict.htm

A Novel Approach for Optimal File Allocation in Memory and Obtaining an Improved Version of Memory Mapping Technique

Jayant Balyan, Khushboo Pahwa and Jyotika Pruthi

Department of Computer Science and Engineering
ITM University, Gurgaon.

Abstract

The imperative nature of memory utilization has always been questioned by the users. Always being under the scrutiny, there have been a lot of attempts for its improvisation. A more common approach is to use memory allocation tables and numerous pointers which require an impermissible usage of memory. The principle objective of this paper is to overcome such limitations through a new vision which considers memory segments in the form of cuboids and their edges representing information such as time and date of their creation. This method can be used effectively in case of sequential access. As a result a simple method for file access with reduced number of pointers is obtained.

1. Introduction

In a computer system, files containing programs and data are organized into a logical structure called the file system. To manage the vast amount of data in a file system effectively, storage hierarchies have been introduced, combining fast but expensive devices with slower and cheaper ones to reduce storage costs and improve access speed. The management of multilevel storage hierarchies deals with the problem of assigning files within a file system to different levels of the hierarchy, based upon expected use. Since it is economically infeasible to store all files on fastest device, files that are used most often should be placed on faster devices, while files that are used infrequently should be placed on slower devices. In this way it is possible for average access time to approach that of the fastest (and most expensive) device while storage cost approaches that of the least expensive and slowest device. File assignment, also

known as file allocation or file loading is the problem of assigning files in a storage hierarchy so as to achieve performance objectives. Changes in file assignment are essential when the usage pattern of the files changes with time and computational loads [1].A data processing system such as a personal computer contains a file allocation table that is stored in memory in a packed format. During initialization, file allocation tables stored on an external storage device are “packed” and stored in a region of memory and subsequently “unpacked” during a read operation. For that purpose, sometimes it feels convenient to use numerous pointers, pointing to different locations of the memory containing data. Regardless to say they acquire some of the memory spaces on their own. This paper concentrates on using the least amount of memory occupied by these clusters of pointers by providing an algorithm which will use the time of creation of any file provided by the operating system to generate its physical address.

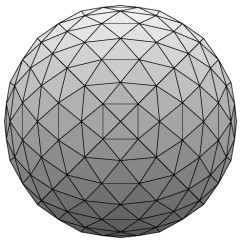


Figure 1: Sphere containing a large number of data segments in the form of cuboids.

2. Approach used

A Cuboid contains twelve edges.
Consider the addressing of each data item.
We can represent each edge with the following way
20.13.08.02.05.12.55.19.09.99
Year : 2013-> 20.13(16bits)
Month -> 8th month i.e. August (4bits)
Week -> 2nd week of month August (4bits)
Day -> 5th day of second week 9th August, Friday (4bits)
Hour of that day -> 12th hour (8bits)
Minute of that hour -> 55rd i.e. 55 minutes past 11 (8bits)
Second of that Minute -> 19th (8bits)
Milliseconds -> 09.99(12 bits)
These together will form 64 bits of addressing.

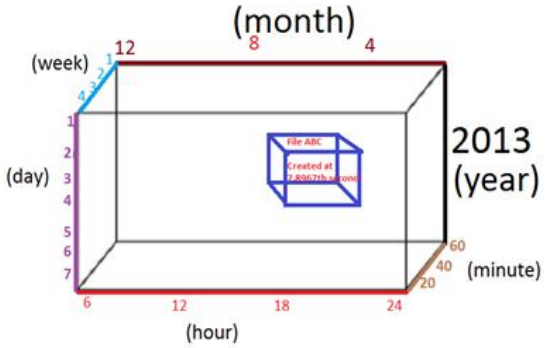


Figure 2: Representing each edge of Cuboid with each of this data.

After searching for the proper cuboid we'll get to know the address and then the data is considered to be stored inside that cuboid or we can say at that memory address. Now, this would be the logical address of the file as shown below.

20.13.08.02.05.12.55.19.09.99

Converting to binary digits

0010.0000.0001.0011.1000.0100.0101.0010.0000.0101.0101.0101.1001.1001.1001.1001

This logical address could be mapped into its physical address sequentially or through some algorithm. For example the physical address of any file created on 2013-08-09 at 6:13pm 30 second 211 milliseconds. Could be let us say,

0010.0000.0001.0011.1000.1001.0101.0010.0000.0101.0101.0101.1001.0010.1010.1010

But its physical address could be:

1000. 0000. 0000. 0100. 0000. 0000. 0000. 0000. 1000. 0010. 0000. 0000. 0000. 0100. 0000. 0001

The algorithm to convert this logical address to physical address has been discussed later in this paper here. Now, these spheres and cuboids are just being used as a way for better understanding, the actual implementation of algorithm may not consider the memory being a sphere or data being stored in cuboids, after implementation the mapping even may look like Figure 3.

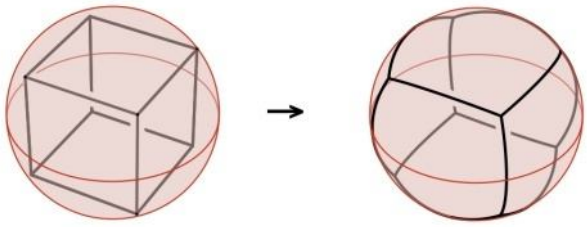


Figure 3: Representation of cuboids inside memory.

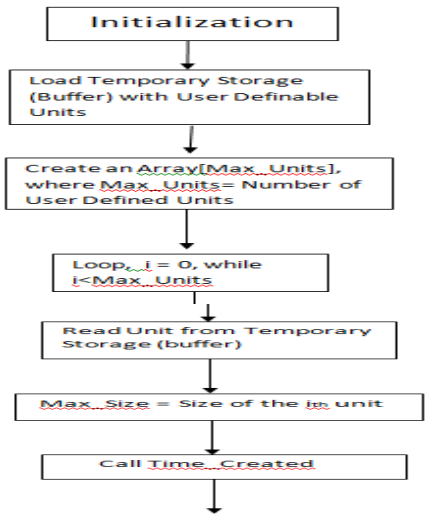
Here, we can see that after the actual implementation of the concept the cuboids inside the memory may look like this.It may be observed how the edges could be deformed to cover the maximum possible area of the sphere i.e. the memory of the system. Thus, here it could be observed that the algorithm must have to deviate a bit from the original concept to achieve the aim of the utilization of the maximum areas of the memory. Subsequently, the edges could be longer or shorter or equal comparing to each other in order to aim for better utilization of the whole memory.

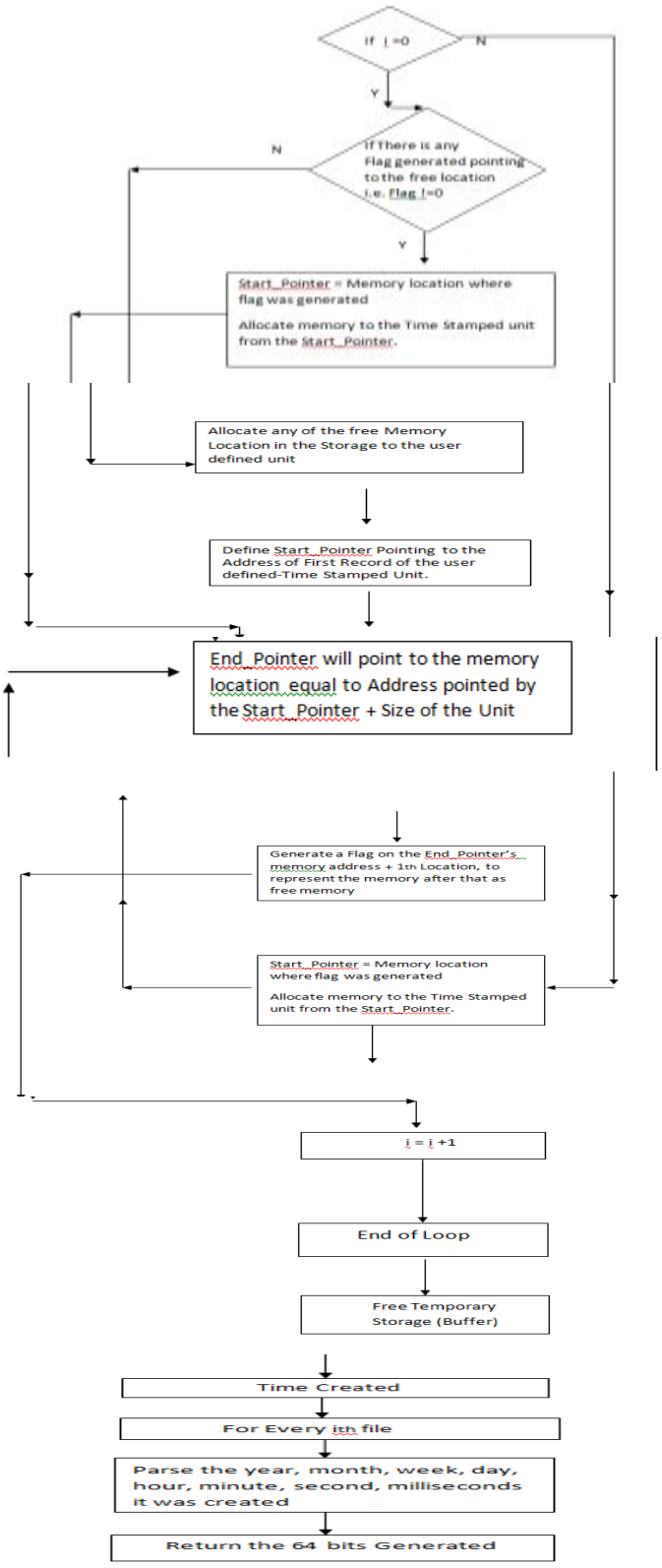
3. Algorithm and method of implementation

To achieve the objective, an algorithm is being constructed to store these files sequentially. Apparently, the need of maintaining tables couldn't be avoided. So, here is the form of table one may get: -

Table I: Table showing time and date of creation of file.

No	**Name**	**Memory allocated (First address)**	**Memory allocated (End address)**
1	20.13.08.04.05.20.55.59.9.9.9	0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000.0000. 0000. 0000. 0000. 0000. 0000. 0001	0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000.0000. 0000. 0000. 0000. 0000. 1001. 1000
2	20.13.08.14.07.24.50.58.9.9.8	0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000.0000. 0000. 0000. 0000. 0000. 1001. 1001	0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000. 0000.0000. 0000. 0000. 0010. 0011. 1001. 1000




If i =0
N
Y
If there is any Flag generated pointing to the free location i.e. Flag !=0
N
Y
Start_Pointer = Memory location where flag was generated
Allocate memory to the Time Stamped unit from the Start_Pointer.
Allocate any of the free Memory Location in the Storage to the user defined unit
Define Start_Pointer Pointing to the Address of First Record of the user defined-Time Stamped Unit.
End_Pointer will point to the memory location equal to Address pointed by the Start_Pointer + Size of the Unit
Generate a Flag on the End_Pointer's memory address + 1th Location, to represent the memory after that as free memory
Start_Pointer = Memory location where flag was generated
Allocate memory to the Time Stamped unit from the Start_Pointer.
i = i +1
End of Loop
Free Temporary Storage (Buffer)
Time Created
For Every ith file
Parse the year, month, week, day, hour, minute, second, milliseconds it was created
Return the 64 bits Generated

4. Conclusion

Rapid increase in memory size has made it difficult to allocate and access memory efficiently. Use of a large number of pointers makes it even more cumbersome. Moreover with the increase in functionalities provided by any system the required for fast access of the user created files has been increased exponentially. Sequential access and allocation has always been the answer to many of these issues. This concept attempts to serve a solution which can sequentially store and access the data and files created by the user.

5. References and Bibliography

[1] Milind B.Deshpande,Richard B.Bunt. Dynamic File Management Techniques http://www.cs.cmu.edu/~sm79/w3/archives/papers/deshpand.pdf

[2] Paul R.Wilson, Mark S.Johnstone, Micheal Neely, and David Boles.Dynamic Storage Allocation :A survey and critical review ,published in Lecture Notes in Computer Science Springer,Volume 986,1995,pp 1-116 http://link.springer.com/chapter/10.1007/3-540-60368-9_19

[3] Bunt,R.B,J.M Murphy and S.Majumdar ,"A measure of Program Locality and its Application", Proceedings ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems ,Cambridge ,Massachusetts, August 1984,28-40.

[4] Chu, W.W and H.Opderbeck,"Program Behaviour and the Page Fault Frequency Replacement Algorithm",Computer,vol 9,No. 11,November 1976,29-38.

[5] Denning,P.J,"The working set model of program behavior",CACM,Vol 11,No 5,May 1968,323-333.

[6] Denning,P.J and Slutz ,D.R,Generalized Working sets for segment reference strings", CACM ,Vol 21,No 9,,September 1978,750-759.