Reason - A Technical Magazine
ISSN 2277–1654
Vol - X ● 2011

General Article

# INSIDE COMPUTATION

**Biswanath Chakraborty**

Department of Mechanical Engineering, Kalyani Govt. Engineering College, Kalyani- 741235
Email: bchakraborti85@gmail.com

**Abstract:** Computation involves both arithmetic calculations and logical operations. Arithmetic calculations can be best described by the operations of addition. In computer science, time taken in executing some elementary operation like addition is taken as one unit. More complex operations like multiplications etc. are assumed to require an integral multiple of this basic unit. On the other hand, logical operations basically involve comparisons which can again be described as the derived conclusion of some arithmetic operations. So, eventually, the question that arises how these computations are done on a modern computer system. The current paper has been focused on this aspect of the computer science.

***Key words:*** Data type, Data Structure, NFA, DFA, Scheduling, Register, Kernel.

## 1. Introduction

It is well known that computer is an electronic device that can perform computations quickly and accurately. It is also known that to get the computation done through computers algorithms are needed which are well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values as output. Thus it may be said that an algorithm is a tool for solving a well-specified computational problem. But the question is how does an algorithm get executed using the hardware? How does the signal, which is generated during the period of execution, activate the hardware?

Basically the systems operate on data and that data interact with other data as per their properties. In order to execute a particular computation, maintaining an interface of a high level language, a proper **data structure** is needed so that the resources can be allocated and locked in time.

## 2. The Role of Data Structure

A data structure can be perceived as an underlying logical and mathematical model which governs the characteristics of a domain. In order to establish data structure as a model, it is required to define the *Domain*, *Function* and *Axiom* (*D.F.A*) of it which will formulate the entire gamut of the data structure.

***Domain:*** Domain may be defined as a *range* of the corresponding data type. The data structure of a data type must hold the values within the range. The domain of a data type (primitive and ADT) can also be defined as a collection of resources. Through this it becomes possible to define which resources can be locked by which data.

***Function:*** Functions define the *type of operations* that can be applied on a particular data, be it of a primitive data type or of an Abstract Data Type (ADT), e.g. the function of '+' may be arithmetic addition if the concerned data are of numeric type (like integer, float, double, etc.), and concatenation if they are of character type.

***Axiom:*** Axioms, on the other hand, define the *set of rules* that govern the functional part of the data structure. For execution of a given user-supplied data, there will be a *definite* (deterministic) and *finite* number of states which may be defined as Q (say). Let the domain (range) be defined as $\Sigma$. Let the functions (i.e. transaction functions) be defined as $\delta$ which is governed by the axioms stated above. Obviously, every transaction has a starting state ($q_0$) and it will generate a definite output at the Final state (F).

Perceiving data structure in this way, it is possible to state that the operation of data structure can be designed in terms of a *Deterministic Finite Automata* (A).

$$A = (Q, \Sigma, \delta, q_0, F).$$

The stratum of data types and their corresponding data structure i.e. the domains, functions and axioms reveals that at the lowest rung there exists the primitive data types advocated by character, integer, float, double etc. These primitives and other non-primitives (e.g. array, structure, class, enumeration, etc.), after a suitable mix, generate the concept of abstract data types and obviously they are composed of a larger domain, a variety of functions which are used for creating a more convenient user-friendly environment [1].

Abstract data types, as the name suggests, encapsulate the heterogeneity which are inherent within them are of two types – linear and non-linear. Here, linearity implies the arrangement either along row or a column e.g. array, linked list, stack, queue. The concept of non-linearity comes into the picture when a breadth as well as a depth of a particular data structure is considered. As a consequence, the traversing techniques like BFS, DFS or some heuristic searches have been evolved.

Data structure, may be linear or non-linear, have some specific usages. In the data structures' stratum, discussed above, graphs have got a niche position as it can depict a real life situation where there are more than one possibilities for traversing a particular *node*. Here 'nodes' represent an object or a variable which are used or locked for writing purposes. Since multiple edges connecting a particular node may generate a Non-deterministic Finite Automata (NFA), a graph may create confusion as to which particular path to be selected. So, the need is to convert it into a corresponding tree where a proper root and the shortest possible paths are selected from the graph traversing among the nodes (NFA to DFA conversion). The tree can be rotated as and when the situation demands and the roots keep on changing. Different techniques are used for this purpose and also different types of trees have been evolved to cater the demands [2].

Non-linearity property of the tree data structure ends up when the nodes are successfully traversed in a particular order (*inorder*, *preorder* and *postorder*) and the nodes are stored in a particular fashion (LIFO) in a typical data structure called Stack. In stack, the nodes are pointed by an arithmetic operation applied through the pointers. Stack pointer increments or decrements with the arithmetic incrementation or decrementation and thus linearity comes in to the picture. The elements are popped off from the stack as the transaction process goes on and is stored in another data structure called Queue where FIFO properties are maintained. It is possible to overrule the FIFO property to make a suitable scheduling algorithm or forecast the best situation by applying some weights on the elements and generating either the corresponding max-priority queue or the min-priority queue.

It is to be noted that arrays and linked lists are the basic building blocks of the advanced data structures. Therefore, queue, in its different manifestation, ultimately decomposed into linked lists and through which the operating systems get the values – *link* different values – *load* them to *relocate* at the proper registers [3][4].

## 3. The Role of the Operating System

An Operating System (OS) could be designed as a huge, jumbled collection of processes without any structure. But this type of design would make it hard to specify code, test and debug a large operating system.

***3.1. Layered approach of the OS:*** Dijkstra advocated the layered approach to lessen the design and implementation complexities of an operating system. The layered approach divides the OS in to several layers. The functions of an operating system are then vertically apportioned into these layers. Each layer has well-defined functionality and input-output interfaces with the two adjacent layers. Typically, the bottom layer interfaces with the machine hardware and the top layer interfaces with users (or operators).

A classic example of the layered approach is the THE operating system which consists of six layers. The **THE multiprogramming system** was a computer operating system designed by a team led by Edsger W. Dijkstra, described in monographs in 1965-66 and published in 1968. Dijkstra never named the system; "THE" is simply the abbreviation of "Technische Hogeschool Eindhoven", then the name (in Dutch) of the Eindhoven University of Technology of the Netherlands. The THE system was

primarily a batch system that supported multitasking; it was not designed as a multi-user operating system. The set of processes in the THE system was static.

Another classic example of this approach is the MULTICS system [5], which is structured as several concentric layers (rings). This approach simplified the complexities related to design and also enhances the protection.

***3.2. The Kernel based approach:*** The Kernel-based design and structure of the operating system was suggested by Brinch Hansen (Fig. 1). The Kernel or *nucleus* is a collection of primitive facilities over which the rest of the OS is built using the functions provided by the Kernel.

Thus the Operating System plays an important role in the execution of the above techniques. The most commonly used modern Kernel based Operating System in the World is WINDOWS. The structure of Windows NT (Fig. 2), depicts that there are mainly two parts of it. The upper part is the User mode (System Interface) and the lower part is the Kernel. The Kernel part is the vital part of the OS in a sense that it is closed to Hardware. Different 'Managers' like Object manager, Process manager, Memory manager, Security manager, Cache manager, PnP manager, Power manager, Config manager, LPC manager are the different authoritative components of the Kernel. They are known as the Executives. Apart from the Managers, Kernel also holds the drivers of the related hardware. The I/O manager provides a framework for managing the I/O devices and provides generic I/O services. It provides the rest of the system with device-independent I/O, calling the appropriate driver to perform physical I/O. It is also home to all the device drivers. The file systems (FAT, NTFS) are technically device driveers under control of the I/O manager. [6]

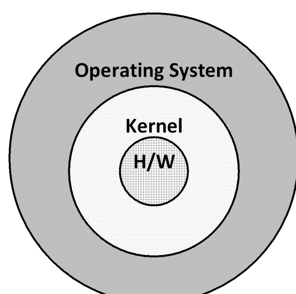


Fig. 1: Structure of a Kernel-based Operating System [5]

## 4. The Influence of C and Low-Level Language

The Executive is written in C and can be ported to new machines with relatively little effort. It consists of 10 components, each of which is just a collection of procedures that work together to accomplish some goal [6].

The users are at liberty to use any language for the creation of different applications. But for execution of those computations, they must pass through the Kernel which is coded in C. Naturally, the rest of the operations will be done through the executives and executives work using the techniques of the 'general purpose' language C. Memory management, process creation, *inter alia*, are all done through C and pointers play the pivotal role in it. Any computation, let it be an addition operation, for the sake of simplicity, is done through pointer operation be it explicitly declared or not.

The simple arithmetic computation of addition may be considered in this case:

```
int  a, b, sum=0;        //Statement 1
a=5;                     //Statement 2
b=3;                     //Statement 3
sum=a+b;                 //Statement 4
```

It is expected that, at run time when the statement 2 is executed, the value 5 will be placed in that memory location reserved for the storage of the value of 'a'. Same thing happens in Statement 1 (sum = 0) and statement 3. In C, a variable is referred such as the integer 'a' as an 'object'.

In a sense, there are two 'values' associated with the object 'a'. One is the value of the integer stored there (5, for example) and the other is the 'value' of the memory location i.e. the address of 'a' in the statement int a=5; 'a' is called lvalue and 5 is called the rvalue. rvalue is stored in the memory address identified by the identifier name lvalue and therefore, 5=a; is illegal [7].

According to Kernigham and Ritchie "An 'object' is a named region of storage; an lvalue is an expression referring to an object" [8].

In statement 4 when int sum=a+b; is written, the previously declared lvalues 'a' and 'b' will be converted to rvalues again through the pointer operation.

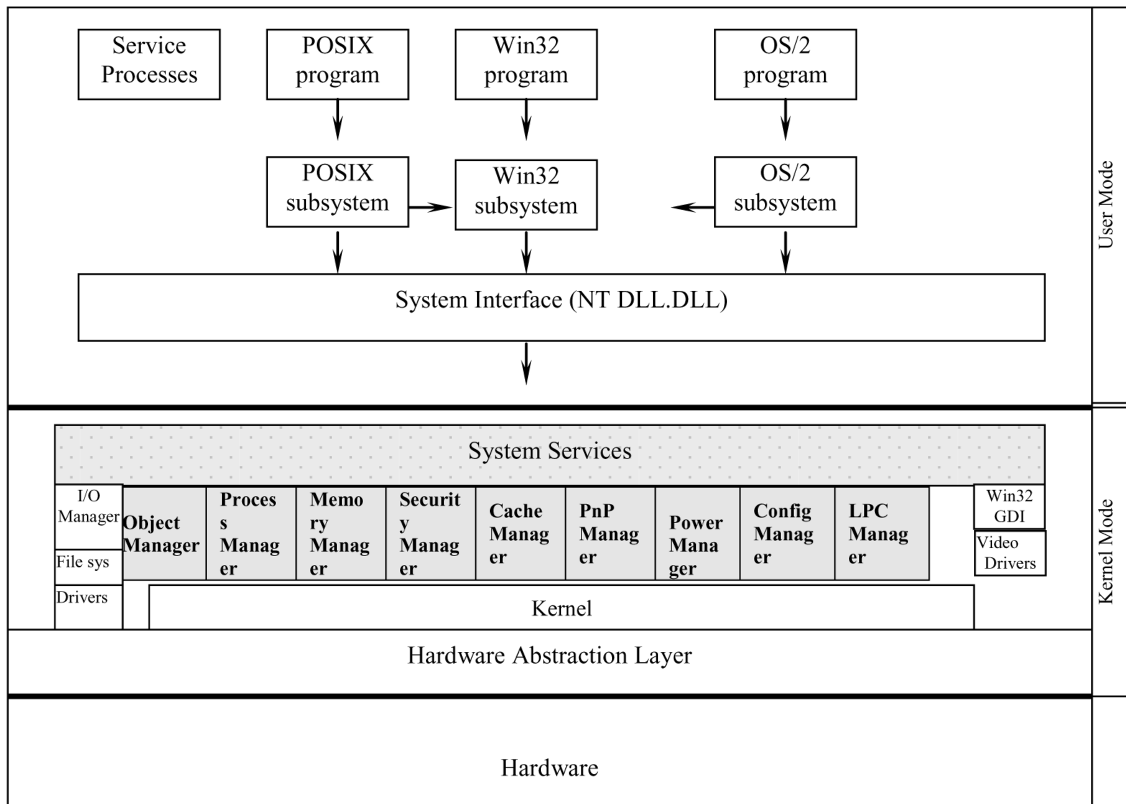


Fig.2: The structure of Windows 2000, shaded area being the Executive [6].

For the execution of the computational process it is necessary to reference a large range of locations in main memory or, for some systems, virtual memory. To achieve this objective, a variety of addressing techniques [9] are employed. They all involve some trade-off between address range and/or addressing flexibility on the one hand and the number of memory references and/or the complexity of address calculation, on the other hand. This time the computational process enters into the Low-Level Language from the High-Level Language.

## 5. Activation of Logic Circuits

Ultimately, it is the time to activate the logic circuits. It is evident from the above discussion that the drivers are to be executed through the OS. The Instruction Set specifically the Operator gets converted into signals through Assembly Level Language [10]. In this case, the Process management of the operating system decomposes the job first into number of sub-processes and each sub-process use different addresses. But when the time of execution comes, these (sub) processes are further decomposed into threads which operate on the single instruction set and activate the corresponding logic circuit. Naturally, they are operated on single address [11].

## 6. Conclusion

Thus, it can be concluded, that behind every computation the pivotal role is played by the data structure. Data structure is a concept and every computation corresponding to any language gets compiled and passes through the pointer management of the general purpose language C. The overall management is governed by the OS. It provides step-by-step aid for memory management, process creation, instruction cycle and fetch cycle management, interrupt handling, virtual to actual memory address mapping and its management. Ultimately, the corresponding logic circuit gets executed and the final output is displayed through the output device.

**References**

[1] Langsam, Y., Augenstein, M.J. and Tenenbaum, A.M. 1996, Data Structures Using C and C++, Second edition, pp.6-14.

[2] Hopcroft J.E., Motwani, R. and Ullman, J.D. 2007, Introduction to Automata Theory Languages, and Computation, pp.55-102.

[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Clifford, S. 2009, Introduction to Algorithms, pp.20-25.

[4] Beck, L.L. and Manjula, D. 2007, System Software-An Introduction to Systems Programming, pp.129-180.

[5] Singhal, M. and Shivaratri, N.G. 2009, Advanced Concepts in Operating Systems, pp.4-10.

[6] Tanenbaum, A.S. 2005, Modern Operating Systems, pp.778-800.

[7] Deitel P.J. and Deitel H.M. 2008, C How to Program, pp.12-15.

[8] Kernighan, B.W. and Ritchie, D.M. 2007, The C Programming Language, p.197.

[9] Stallings, W. 2003, Computer Organization & Architecture-Designing for performance, pp.382-388.

[10] Gaonkar, R. 2009, Microprocessor Architecture, Programming and Application with the 8085, Fifth Edition, p.46.

[11] Deitel H.M., Deitel P.J. and Choffnes D.R. 2007, Operating Systems, pp.110-181.