MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 92

The Current Challenge to Computer System Architects*

by

Jack B. Dennis

*Outline of a talk given at the International Workshop on Computer Architecture, Grenoble, France, June 26-28, 1973

This work was supported in part by the National Science Foundation under research grant GJ-34671.

October 1973

THE CURRENT CHALLENGE TO COMPUTER SYSTEM ARCHITECTS

JACK B. DENNIS
Project MAC, MIT
Cambridge, Mass. U.S.A.

THEME

Two circumstances are holding back needed advances in the organization of general purpose computer systems:

The desire to maintain compatible implementation of programming languages that embody obsolete constructs.

The reluctance of computer architects to adopt a view of memory better suited to the data representation needs of contemporary computer applications than the ordered array of words.

Summary

Manufacturers of general-purpose computer systems face a difficult challenge in meeting foreseeable user demands.

The ability to correctly execute existing application software is always a consideration of high priority to the manufacturer.

Nevertheless, the market demands the introduction of new capabilities for security, data base management, sharing of information, and any capabilities that lead to reduced costs of program construction.

Modularity - the ability of a computer to support the construction of new programs from independently written program modules - is a potentially powerful approach to reduction of program development costs.

Computer systems that meet requirements for security and modular programming must be based on concepts of memory organization appropriate to the abstract data structures of modern programming practice.

The prospect of very inexpensive memory and processing hardware makes it attractive to apply hardware to improved computer architectures for the support of security and modular programming capabilities.

We point out that providing the desired new capabilities conflicts with providing transferability for existing application software.

In particular, certain aspects of widely used programming languages (for example, pointer data in PL/1) depend on relationships of addresses in a conventional memory array.

On the other hand, modular programming and security of information require that a tree-structured view of memory be embodied in the computer system architecture.

The resolution of this conflict is a revision of language definition to be consistent with the requirements of modular programming and security of information.

Successful development of computer systems that provide the desired new capabilities must make use of precise semantic models for their functional behavior.

1. The manufacturers' goals

Provide implementation of Cobol, Fortran, PL/1, consistent with accepted standards for these languages.

Meet de facto standards for important data base access methods.

Provide hardware and configuration independence for programs and data. Provide transferability of programs and data between systems.

Permit communities of computer system users to share procedures and data bases.

Ensure that access to procedures and data bases is permitted only where duly authorized (security).

Apply advantageously the decreasing cost of memory and data processing hardware.

Provide means for easier program construction.

2. Modularity

The modular construction of programs is potentially a powerful means for reducing the effort required to obtain correct application software.

A computer system supports modular programming to the extent it permits new program modules to be formed by combining existing independently specified modules without knowledge of their internal construction.

Modular programming has strong implications with respect to computer system design:

1. Uniform representation for all data types to be communicated between program modules;
2. Sufficiently general conventions for interfacing program modules;
3. Ability of a program module to create, access and communicate data structures of arbitrary complexity.

The requirements of modular programming become more difficult to meet as the modules depend on more advanced data structures and programming constructs.

The ideas of modular program construction and of "structured programming" are closely related and have similar implications regarding the design of programming languages and computer systems.

3. Security

Security becomes an important design objective when a computer system serves the needs of many users and users are able to share programs and data bases.

In a secure system access by a procedure is permitted by the system only to those procedures and data structures for which authorization has been given by the owner.

Some issues concerning security are: What are the kinds of objects to which authorization pertains? What are the entities that grant and receive authorization? By what mechanism is authorization accomplished?

A satisfactory security scheme must be hierarchical in the sense that access to a data structure or procedure implies access to component data structures and procedures. This is required by modular programming. It must be possible to authorize access to a compound object in one step.

The objects to which authorization applies should be program modules (procedures) and data structures.

4. Evolution of the virtual memory concept

Program modularity requires that the computer system implement a uniform scheme for accessing information regardless of the physical medium in which the information resides. The concept of automatic page replacement was a significant initial step.

The Burroughs B5000 computer system introduced the segment as a unit of information on which modular programming and security is based.

In Multics the concept of virtual memory has been advanced so that all on-line procedures and data structures of a large community of users may be accessed in a uniform way. Any procedure or file residing in the Multics file system may be shared by any number of users once authorization of the owner has been obtained.

These schemes are important in computer system design. Nevertheless, these designs are ad hoc - an understanding of their foundation in precise semantic terms is missing. Moreover, the information unit (the segment) is not a natural information unit for representing many important information structures of source languages.

5. Language-based views

Iliffe's concepts in Basic Machine Principles show how the organization of computer systems may better reflect the requirements of programs using dynamic data structures. This illustrates that important data structures do not map efficiently onto conventional memory.

The Record Handling constructs of Hoare show how security may be obtained for single programs by an elegant language compiled for a conventional machine. Yet this work does not show how to realize a secure multi-program system that supports modular programming.

6. The Design of a base language

A new approach to the achievement of modular programming and security is needed based on the rational design of the base language to be realized by the computer system.

We require:

1. A uniform representation for elementary objects: integers, reals, strings;
2. A uniform scheme for constructing and accessing data structures built from elementary objects;
3. A suitable notion of program module.

We are considering the design of a base language using rooted acyclic directed graphs as the fundamental objects for representing data structures and program modules. We associate selectors (integers or strings) with the branches of an object and elementary objects with the leaves.

A program module is an object that defines a transformation of objects into objects.

We let the allocation of storage for data elements (nodes and leaves of objects) be managed by the "reference count" method so elements that become inaccessible are deallocated and their storage freed.

We have shown that many important constructs of contemporary source programming languages can be effectively translated into such a base language.

7. Translatability

The computer system architect must demonstrate that his proposed system is able to support effective translation for essential source languages.

If the computer system design retains all constructs (such as memory as an indexed array and sequential instruction execution) on which earlier language translators are based, then consistent translators are easily constructed for the new system.

If the conventional notion of memory is not supported by the new system, the system architect is obliged to prove that an effective and correct translator can be built.

The only proof acceptable at present is demonstration of an actual translator using simulation of the proposed system.