MCKENDRY84

McKendry, Martin S. and Campbell, Roy H. Implementing Language Support in High-Level Languages
IEEE Transactions on Software Engineering, Vol. SE-10
No. 3 (May 1984) pp. 227-236

One of the requirements for building an operating system in a high-level operating system language such as Ada, Modula-2, or Concurrent Pascal is a construction of a kernel (language support system). This paper presents a model that generalizes the concept of a kernel and the processes it supports to be at different levels of abstraction.

Many high-level operating system languages are dependent on run-time kernels that are programmed in assembly language, even though most of the logic implemented in run-time systems is expressible in existing high-level languages. The primary reason, authors argue, run-time systems(kernels) are coded in assembly language is the inability of existing high-level languages to express the interactions between run-time systems and the high-level languages they support. The lack of facilities that would enable high-level languages to express a mapping directly onto hardware is a less significant problem that can be easily solved.

Kernels support operating system language features that require manipulations of a process' environment that cannot be expressed in conventional languages. What is lacking is a means for the kernel to recognize requests from processes and from hardware and a means for the kernel to recognize requests from processes and from hardware, and a means for the kernel to assign a physical processor to a data structure that represents a process. The kernel also requires some mechanism to access the parameters to requests. A language mechanism is proposed to provide these functions so that an entire kernel, together with its relationship to the processes it supports, could be programmed in a high-level language.

A new mechanism called the Execute statement is introduced to express the interface between a process that implements run-time support, and the high-level processes it supports. The Execute statement enables a process (written in a high-level language) which provides support functions to switch the processor to processes (also written in a high-level lanaguage) that use those functions.

The authors go on to present a model of operating system structure and a model of capability access. As well, they deal with the safety characteristics of high-level languages and suggest safety problems which might arise in the design of a context switching mechanism.

Pascal is used as a basis for development of Software capabilities which control access between levels of abstraction while the Execute statement controls processor switching between levels. The mechanisms presented rely on data typing for reliability and protection. They encourage systems that are well protected and exhibit an explicit hierarchical structure.

After preliminary discussions the functionality of the Execute statement and the access it permits between levels of abstraction is examined, and a pilot implementation on a Prime 650 computer using Path Pascal high-level language is discussed.

The article ends with the discussion of the experimental operating system encompassing the use of software capabilities and Execute statement. The authors also suggest several extensions to the mechanism implemented, notably extensions to manage interrupts, timeslicing, and preemption. A new level of abstraction may have to be introduced into the model to handle multiprocessing environment.

This article is well written and a structured exposition of the material presented in it aids in the ease of its assimilation.