Second Edition

OPERATING SYSTEMS PRINCIPLES

Stanley A. Kurzban
Thomas S. Heines
Anthony P. Sayers

OPERATING SYSTEMS PRINCIPLES

SECOND EDITION

Stanley A. Kurzban
Thomas S. Heines
Anthony P. Sayers

VAN NOSTRAND REINHOLD COMPANY
New York

Contents

1
Introduction

The term "operating system" came into widespread use in the late 1950s. Sayers [Say71] defines an operating system as "a set of programs and routines which guide a computer in the performance of its tasks and assist the programs (and programmers) with certain supporting functions." This definition is accurate and useful, but by no means the only one. The American National Standard definition is: "Software which controls the execution of computer programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management, and related services." This definition, while similar to Sayers', seems too restrictive and too dependent on other terms which themselves are jargon and require definition.

We can expand Sayers' definition by naming the supporting functions provided or the programs assisted. Alternatively, we may make reference to current practices in the computer industry by defining operating systems as "that programming which is provided by the vendor of a computing system as an integral part of the product he markets." This, of course, yields an incomplete and imprecise definition which varies with vendors and computers.

Let us, then, accept Sayers' definition of an operating system with some qualifying remarks. Some authorities would restrict the term "operating system" to a set of programs which creates an apparent computing system as comprehensive as the original hardware system, but simpler and easier to use. For our purposes, however, those programs which are peripheral to the computing system, but vital to the effective use of the system and of potential value to all those who use the system, are included. Examples of such programs are utilities (for example, file-copying programs, sorts, and listing programs), spooling[1] programs, and routines for managing networks of computers. Language processing programs – assemblers, compilers, and interpreters – are considered here only insofar as they are influenced by the characteristics of particular operating systems.

Having defined our subject, we next consider how operating systems have evolved to their current level of sophistication. We then discuss the purposes

---

[1] Spooling, from *s*imultaneous *p*eripheral *o*perations *on-l*ine, is the processing of data between a device designed for communication with people (a printer or card punch, for example) and an intermediate storage device (such as a disk or magnetic tape device).

served by operating systems, their constituent parts, some programming techniques used in their development, and the influence of computers upon the operating systems which support them. These topics will lay the foundation needed for more detailed study of operating systems.

1.1 HISTORICAL PERSPECTIVE

The development of operating systems has been marked by two different types of progress: evolution and retrenchment. Within generations of computers, evolution has dominated, with successive advances occurring in response to problems successively encountered. The problems often arose as a result of earlier advances, a phenomenon Marine [Mar70], in another context, attributes to "the engineering mentality," the way of thinking which insists upon solving every problem individually without regard for related problems or side effects. Such an approach, despite its seeming inadequacy, yields prompt solutions to problems, a requirement in the fast-advancing field of electronic computing. This type of development has fortunately been reinforced by retrenchment, more comprehensive advances, which consolidated earlier technological gains. These advances have most often coincided with the introduction of radically different computers, but also at times, for combinations of less than obvious reasons, occurred in the absence of hardware innovation. Radically new operating systems, not truly revolutionary, but significantly innovative, have permitted the consolidation of earlier advances and the development of integrated sets of new programs which provide totally new facilities.

Before there was an operating system, there were computers. With little or no accompanying programming, the computer was a very complex tool, difficult for even its designers to use efficiently. Systems with one-card loaders and primitive assemblers could be of use to those with the patience and analytical skill needed to break a problem down to a succession of additions, divisions, etc., provided someone well acquainted with the computer could write input/output (I/O) routines for them. But some systems developed prior to 1955 lacked even an assembler. They had to be programmed in octal or decimal, with the programmers supplying operation codes and branch locations by themselves. The addition of a single instruction near the beginning of a program meant the relocation (and repunching) of perhaps thousands of addresses, or patching – the addition of code at the end of a program and the replacement of an existing instruction by a branch to the new code. The new code had to begin with the replaced instruction, and end with a branch back to the location following the replacement. A program with many patches was not only inefficient, but also very difficult to debug.

The users of these primitive systems normally operated the computer personally. The absence of both aids for debugging and established operating procedures