CGTM # 43
E. Satterthwaite
March 1968

Notes on Construction of Subsystems within Operating System/360

1. Introduction and Summary

According to its designers, Operating System/360 (ØS) is "one of the first 'second generation' operating systems", and it attempts to "accomodate an environment of diverse applications and operating modes"[1]. This note is an attempt to summarize some initial experiences with ØS, particularly some problems encountered in the design and implementation of a batch processing subsystem operating within the framework of ØS. Section 2 is a brief description of the design and history of that subsystem. In section 3, an attempt is made to justify the construction of such subsystems. Some specific shortcomings, relative to the implementation of subsystems, of current versions of ØS are presented in section 4. Finally, Section 5 briefly summarizes the concept of hierarchy of control and suggests that the designers of ØS failed to provide for such a hierarchy although adequate facilities could have been implemented fairly easily.

2. The PL360/ØS Subsystem [2]

During late 1965 a project to design and implement a programming system for the IBM System/360 series of computers was undertaken by Niklaus Wirth and co-workers at Stanford. That programming system was based on two languages:

(1) PL360, a language with ALGOL-like structure which provides (and requires) the specification of machine operations at the level of a conventional assembly language [3];

(2) ALGOL W, a language which is approximately an extension of ALGOL 60 to allow general plex processing based on the concepts of records and references [4].

Both of these languages were designed to be suitable for instructional use and for research in programming techniques. (To date, the major application of PL360 has been the construction of experimental compilers, interpreters, and editors, while ALGOL W has been used primarily for teaching introductory programming and for research in numerical analysis.) Thus considerable emphasis was placed on providing efficient compilation and meaningful diagnostic error messages.

The early releases of ØS available during the initial phases of this project were totally unreliable, and the standard system then in use on the available hardware was a very unsatisfactory version of BPS. It was therefore decided to implement a stand-alone system operating completely independently of any IBM software; the PL360 compiler was, in fact, originally written in Burroughs B5500 ALGOL. By early 1967, the PL360 compiler and several utility programs had been completed and were being heavily used in writing the ALGOL W

compiler, substantial parts of which had also been completed. It was then becoming evident that operating time on the various Stanford computers for a non-standard system would become increasingly difficult to obtain. Thus it was decided to make PL360 and ALGOL W available within a standard ØS environment. In providing such availability, satisfaction of the following three constraints was considered essential:

(1) the ØS versions should be entirely compatible with the stand-alone versions at the source language level (including I/Ø capabilities);

(2) the ØS interface should not substantially degrade the efficiency of the programming system within its intended range of use;

(3) no non-standard or version-dependent features of ØS should be used.

Certain conventions which had been established for PL360 and were reflected in the language itself were incompatible with corresponding ØS conventions; thus PL360 could not easily be modified to produce ØS object modules without obsoleting much prior work. In addition, it was then obvious that ØS job-to-job and linkage editing overhead would unacceptably degrade the performance of the system. Thus the most attractive alternative was the development of a closed subsystem within which PL360 and ALGOL W jobs could be batched.

3. The Justification of Subsystems

As noted, the primary objective in the design of ØS was to provide for diverse applications and operating modes, i.e., it was intended to be a very general system. Programs written in the ØS Job Control Language may be considered directives specifying the effective creation of a virtual machine which includes specified data sets and processing operations. The properties and interrelations of these data sets and processing operations may vary over a substantial range. Experience has shown that the facilities of ØS do provide fairly powerful and general tools which are of considerable value in performing sufficiently complex and structured programming tasks. Obtaining such power and generality has proven, not surprisingly, to be quite expensive. In particular, a primary technique used in the implementation of such facilities has been the provision of "parametric generality" [1]. A large number of parameters can be, and often must be, supplied at each level of interface with ØS (e.g., Job Control Language, language processors, control program services, etc.). It has developed that most of these parameter specifications are essentially program strings to be interpretively executed, and that execution often invokes the fetch of programs from auxiliary storage.

The user to whom PL360 and ALGOL W are primarily directed pays two heavy penalties for the use of standard ØS language processing facilities:

(1) The computational complexity and expense of creating the required virtual machine configuration(s) exceed those of the user's computational task itself, often by an order of magnitude.

(2) The user is required to learn some part of the jargon for specification of such virtual machines (e.g., Job Control Language), and he also

> receives cryptic, machine-and system-oriented error messages. The user generally has no interest in learning any of these areas, nor are they currently documented in a manner intelligible without substantial effort.

Some of the above remarks are, of course, criticisms of the design goals, implementation, and documentation of ØS, and one hopes that each area will be carefully considered and improved in future operating systems. Both IBM and most larger System/360 installations are, however, committed to ØS in approximately its present form. Thus the problem of providing the smaller user with convenient and economical programming tools within ØS is of considerable practical importance.

One possible solution is the construction of subsystems which process as a single ØS job step a batch of smaller jobs under the direction of non-ØS control statements. A suitable virtual machine need be created only once, so that most ØS overhead (JCL processing, opening data sets, most resource allocation, etc.) is the incurred only once per batch. In addition, the average user need only be aware of the (generally much simpler) control language of the subsystem. Finally, with a reasonable hierarchy of control, the subsystem should be able to detect error conditions and provide a message related to the user's source language and program.

4. Difficulties Encountered

In any truly satisfactory subsystem of the type described above, the following requirement must be met:

> No program executed under the subsystem can prevent the execution of, or influence the results obtained from, any other independent program so executed.

(Programs which utilize common data on input/output devices are not considered independent in the above sense.) This requirement has several implications:

(1) Partitioning of an allocated resource (such as time) must be under the control of the subsystem executive;

(2) Each program must be unable to destroy or alter in an unanticipated way any part of the subsystem monitor;

(3) The superior executive must allow the subsystem executive to diagnose and recover from any possible error in any program which it processes.

It should be noted that PL360 is especially demanding in the latter area. Although designed to encourage good programming practice and reduce certain common errors, it allows the programmer to direct that any arbitrary bit pattern be executed. In addition, the provision of input/output facilities sufficient for writing file maintenance programs also provides an indirect facility for specifying the execution of a wide range of unacceptable channel programs. Thus it is impossible for the PL360 compiler to perform sufficient validity checking to prevent fault conditions.

The PL360/ALGOL W subsystem anticipates the following program errors:

(1) program interrupts

(2) invalid input/output requests

(3) input/output operation errors

(4) time limit violations

In each case, the subsystem executive attempts to provide a meaningful diagnostic message and to continue processing with the next program in the batch. ØS provides no facility for dealing with two more errors occasionally encountered:

(5) destruction of part of the subsystem executive

(6) invocation of ØS system services (execution of SVC instructions) with improper parameters.

It was somewhat surprising to discover that, despite the multitude of facilities offered by ØS, only the first type of error can be processed in an entirely satisfactory manner. In the following paragraphs, particular errors and error analysis facilities are considered.

4.1. Program Interruptions

The program interrupt processing facilities provided by ØS are well-designed and hence will be described in some detail. By use of the SPIE SVC instruction [5, pp. 38-40], the problem program can specify an exit routine for processing selected program interrupt conditions. Those not selected for such processing are handled by the ØS executive in a standard way (termination of the job). In the exit routine, the PSW and general register contents existing at the point of interruption are available for analysis. In addition, the register contents and the low order word (program resumption address) of the old PSW may be altered before return. It is slightly inconvenient but understandable that control must be returned to the ØS executive before processing continues. If a program check occurs within the exit routine, ØS terminates the problem program.

With these facilities, the subsystem executive is able to locate the offending instruction, issue an appropriate message, and then specify that execution resume with the subsystem dispatcher rather than the faulty program. In addition, protection is automatically provided against an infinite loop caused by an untested analysis routine generating new errors of the same type.

4.2 Input/Output Operations

Before the use of any data set, it must be opened by use of the OPEN SVC instruction [5, p. 78]. For a variety of reasons, opening may be impossible. In some cases, such as the provision of no corresponding DD statement, the open function is simply considered unsuccessful and control is returned to the calling program. That program may detect such a condition by testing a suitable bit [6, p. 121], issue an appropriate message, and continue other processing.

Various other conditions, such as input/output errors or label verification errors (see completion codes 113, 213, ..., F13 of [7]), however, cause the entire job to be abnormally terminated.

Actual input/output operations are initiated and controlled through an elaborate set of system control blocks. In one of these, the data control block, provision is made for the specification of two exceptional exits [5, pp. 65-75]. One of them, called the EØDAD routine, is entered when an attempt is made to read beyond the end of a data set. The other is called the SYNAD routine and is given control by the ØS executive to analyze and act upon "exceptional conditions or uncorrectable errors" in input/output operations. For the SYNAD routine, a pointer is provided to the relevant channel and device status bits, and in both cases an appropriate message can be provided and processing continued. In practice, however, it appears that control is passed to the SYNAD routine only in the case of a unit check condition. Other exceptional conditions, such as exhausting available space, attempting to write a non-overflow record exceeding track capacity, and using an invalid data address, cause termination of the entire job (see completion codes 002, 100, 400, 800, B63, FIF, etc., of [7]).

Thus in the case of some input/output specification or execution errors, the subsystem executive is given no facility for recovery or diagnosis, while for other similar errors an adequate facility is provided.

4.3 Timing Interruptions

By use of the the STIMER SVC instruction [5, pp. 36-38], a problem program can specify a time interval and an exit routine to be given control upon the expiration of that interval. In marked contrast to the program interruption exit routine, that exit routine appears to be useless except for polling applications. There is no method specified for examining or altering the PSW and general register contents existing at the point of interruption (or at the last time the task was active). IBM suggests ([5], p. 38, example 22) that the exit routine set a flag to be subsequently polled by the executing program. This solution is adequate only for the least important possible case, however; for if the programmer anticipates that a loop may fail to terminate, other (and usually better) tests are available. In the more relevant case in which the loop was not anticipated, polling is useless.

The PL360/OS subsystem does, in fact, provide a reasonable timer trap and diagnostic, but only by some rather pornographic trickery (which was adapted from Dave Wortman's SPL/XPL Subsystem). Essentially this trickery involves tracing system control block pointers, in a manner not described (or guaranteed) in any generally available IBM literature, to locate the old PSW. Then, except in some special cases, the operation code of the instruction at the PSW return address is made invalid, and control is returned to the system to force an eventual program interrupt.

4.4 SVC Interruptions

ØS provides no method for a task to specify selective examination or processing of SVC interruptions associated with that task. It is possible for PL360 programs to execute SVC instructions requesting ØS control program services. In such cases , the parameter registers are usually incorrect; the results tend to be both spectacular and fatal to the job step(or job in MFT or PCP).

4.5 Memory Protection

At present, all memory available to a job is assigned a single protection key, and that key appears in the PSW of every task associated with that job. In particular, user programs are able to write into the subsystem executive. Most accidental instances of this could be prevented by setting the protection key of the memory area containing the subsystem executive to zero during execution of user programs (thus making the subsystem read-only to the executing program) and resetting it to the job key at the end of such execution, but ØS provides no facility for doing so.

5. Hierarchy of Control

During the processing of a user program in the PL360/OS subsystem, one can distinguish three levels of executive control: the ØS executive, the subsystem executive, and the user program (usually with a trivial executive function). Preceding sections have argued that all exceptional conditions which can arise at the level of the user program must be diagnosed and rectified by the subsystem executive. They have also shown that current versions of ØS provide an incomplete set of facilities for doing so. It is fairly easy to suggest ad hoc modifications of ØS to provide such facilities, and the economics of servicing users with relatively small computational tasks suggests that such modifications be made. Specifically, the following changes would be desirable:

(1) Information similar to that provided in the program interrupt exit routine should be provided in the timer exit routine, as should the ability to modify registers and the return point. This information is certainly available in the system, and could be provided in a manner completely compatible with current specifications for the timer exit.

(2) The programmer should be given the option of specifying in a data control block that control be returned unconditionally after an attempt to open, close, or transfer data to or from a data set. If such an option is selected, information currently supplied in the system completion code should be available for analysis. It might, for example, be returned as a register 15 completion code for OPEN and CLOSE and in the status indicator area [6, pp. 263-264] for services providing for entry to the SYNAD routine. This modification should also be relatively easy to implement and compatible with current programming.

(3) It would be very desirable to provide a pair of SVC instructions which, after appropriate validity checking, switch the protection key of a part of a partition (region) between zero and the key associated with the corresponding job. It appears that such a set of instructions cannot even be safely supplied by the installation, for the relevant program logic manuals suggest that in PCP and MFT storage keys are set only upon initial loading of OS; in particular, they are not reset after abnormal job termination. Provision of such a facility would probably require substantial modification of those versions of OS.

(4) An even more substantial modification of OS would be the provision of an SVC interrupt mask similar to the program interrupt mask currently provided. Such a mask should enable a specified exit routine to conditionally inhibit the execution of selected supervisor services requested by the task. Such a facility is necessary for completely effective user of the memory protection scheme discussed above. It is, however, probably not practically implementable for at least the following reasons:

   (a) it would require substantial modification of the OS executive;

   (b) the additional overhead would probably be prohibitive in certain critical applications (especially input/output synchronized with a mechanical device);

   (c) Careful analysis of the ØS control program would be required to insure that inhibition of certain SVC's associated with a particular task (such as WAIT and PØST) could not cause race or lock-up conditions not local to that task.

It is possible to extend the idea of a hierarchy of control to an arbitrary number of executive levels; in fact, the ALGOL W control routines could have been implemented naturally as such an additional level. The preceding discussions indicate that the executive of each level should be able to deal with all exceptional conditions generated or passed by the immediately controlled level. Such an idea is, of course, not new; Dennis and Van Horn's spheres of protection and associated capability lists [8] include approximately the same ideas. Dennis and Van Horn were motivated by the problem of debugging programs in a complex environment, while the motivation of the present work was the provision of economical and reliable subsystems within a complex operating system. If current trends in operating systems and programming environments continue, it will be essential for future hardware and software to provide for such a hierarchy of control in an efficient and uniform manner.

REFERENCES

[1] G.H. Mealy, "The functional structure of OS/360", IBM Systems Journal 5 (1966), pp. 3-11.

[2] N. Wirth, J. Wells, and E. Satterthwaite, "The PL360 system", Technical Report CS 91, Stanford University, February 1968.

[3] N. Wirth, "A programming language for the 360 computers", Technical Report CS53 (Revised), Stanford University, June 1967.

[4] H. Bauer, S. Becker, and S. Graham, "ALGOL W", Technical Report CS89, Stanford University, January 1968.

[5] IBM System/360 Operating System Supervisor and Data Management Services, IBM Systems Reference Library Form C28-6646.

[6] IBM System/360 Operating System Supervisor and Data Management Macro-Instructions, IBM Systems Reference Library Form C28-6647.

[7] IBM System/360 Operating System Messages, Completion Codes, and Storage Dumps, IBM Systems Reference Library Form C28-6631.

[8] J.B. Dennis and E.C. Van Horn, "Programming semantics for multiprogrammed computations," Comm. ACM 9 (March 1966), pp, 143-155.