What is the use of operating systems?

By P. L. Cloot*

Before long nearly all computer users will be using operating systems with their equipment. These operating systems consist of both language processors and supervisor programs, and this paper sets out to explain in simple terms why the need for such systems has arisen.

The aim of this paper is to show that computer operating systems are, from the user's point of view, not only inevitable but desirable. It is not intended to be a technical account of operating systems or of particular ways of implementing them, as enough papers and manuals are already available on the subject. The proceedings of the Edinburgh Joint Computer Conference (1964) give a broad survey of users' views on computers, and a more specialized book has recently been published (Wegner, 1964) which represents current thinking in this country on operating systems. It is hoped rather that this paper will serve as an introduction to the subject for those who need to know what all the fuss is about.

Another area it is not proposed to tackle is that concerning the physical equipment actually used to implement any particular machine system. Endless arguments can be, and often are, carried on about the relative merits of various forms of "hardware." All the user is concerned about is how the overall system behaves, and whether the hardware the designers chose to implement can be manufactured punctually to be reliable at a reasonable price. (It is blithely assumed in saying this that the user is confident of his ability to make effective use of the equipment if it is installed in working order on time! This raises a whole host of other problems, which again are not the subject of this paper.)

Problems in using a computer

In order to use a computer it is necessary to have a language for communicating intelligence to it and a means of ensuring that a program written in this language functions as intended. Let us consider programming languages and program testing, and then see what other problems in using a computer arise from them.

Programming languages

It is generally recognized, except in a few last pockets of resistance, that there is something to be gained from the use of languages of higher level than those of the computers themselves. The programmer's time saved in writing and correcting a program far outweighs any additional computer time used in language translation. If, in addition, the resulting reduction in program testing time is taken into account, there may even be a reduction in total computer time used. In fairness to those remaining pockets of machine-language programmers, the blame should probably be placed on the computer designers, since they have either failed to produce adequate language processors or have failed to enlighten the users as to their advantages. It is disturbing to find coming from one of our leading mathematical laboratories such an understatement as this: "A computer which obeys, say, five instructions per microsecond is probably going to consume programs faster than any reasonable team of programmers can produce them, if they have to work in machine code." (Barron and Hartley, 1964.) It is, of course, unfair to take this quotation away from its context, and the authors do go on to say later: "It is time that software (operating systems as well as compilers) and hardware were treated as equal contributors to the total problem-solving objective of computers, . . ." Even in this country the ratio of the cost of programmers' time to the cost of computer time is increasing, and users will eventually be forced to realize that it may pay them to improve the efficiency of their staff, if necessary at the expense of computer time. We may conclude that high-level languages, with their processors and their diagnostics, are here to stay.

It is also necessary to realize that a language processor, which is a complicated program to translate a high-level language into machine language, must reach a certain level of efficiency, in spite of the preceding comments on the greater importance of the efficient use of programmers. In the past, some processors have not done this, and in particular have not had adequate diagnostic facilities built into them. Errors in programs can be classified into two groups: clerical errors, where the programmer has instructed the machine to do something which is physically impossible; and logical errors, where the machine has been asked to do something which is valid, but not what the man with the problem intended. In the first group the computer designer has an obligation to learn from users' experience what errors are made and to ensure that these are diagnosed by the processor. (To take an obvious example it is inadequate for a processor to desist from translation when it has found a single error in a program: it should attempt to find as many as possible each time it is used.) The second group is no concern of the processor but comes in the area of the external system design and control. The designer can still help here by means of aids to dynamic program testing.

* *IBM United Kingdom Limited, Education Centre,* 15–17 *Lodge Road, London, N.W.*8.

Program testing

In the best planned installations, even those expecting to do entirely routine commercial work, programs do have to be tested occasionally. If an organization makes no changes in its requirements for too long, it is probably an indication that its management is moribund. And even the best of managers make mistakes or change their minds once in a while, so modifications will undoubtedly be required sooner or later.

Developing, modifying and testing programs involves the use of these complicated language processors for short periods, so even the installations with a large, stable workload are bound to have a variety of what amount to short, complicated jobs to run. For installations where program development is the order of the day, the problem of handling a large number of very short jobs rapidly becomes acute.

As already mentioned the effectiveness of the individual test runs can be greatly increased by an efficient dynamic testing system which enables a programmer to perform dumps of selected quantities at selected stages in the execution of a program, to help deduce where his logical errors lie. This again is an area in which the designer can profit greatly from users' feedback in deciding what form of tracing or dumping facilities are going to be most helpful.

While discussing short jobs, it is probably appropriate to remember that there are many new short jobs which would arise spontaneously, if an adequate means of handling them were available. This trend is desirable, as long as these new jobs are economically worth while. It might be as well just to bear in mind Parkinson's law about the expansion of work to occupy the computer time available.

Supervisor programs

Having established that all installations will have some short jobs to process, and many will find that perhaps a half of their total running time is composed of short jobs, the question of job set-up time and transition between jobs becomes important. This is again an area where the designer has an obligation to help, and most users are beginning to accept that they will profit by allowing the computer to control the transition between jobs. This is done by means of a "supervisor program," a part of which will always reside in the high-speed store of the machine. By the use of such programs, a stack of jobs can be given to the machine, separated by control information for the supervisor to recognize what type of job follows. The time saved overall should more than compensate for the space occupied by the supervisor in all but exceptional circumstances. The supervisor program in practice has many other additional functions to perform, and these will be referred to later.

The use of a supervisor implies that the programmer is no longer allowed anywhere near the operator's console, and some of the old school may resent this. However, forcing programmers to do their thinking away from the machine can do nothing but good, and may even encourage them to look ahead and anticipate problems which may arise in testing. No one these days would dream of installing a complete computer system without first doing a comprehensive systems analysis and anticipating the problems which its installation might cause. In the same way, but at a much lower level, no programmer should be writing a program without having thought of the implications and possible difficulties in the implementation of his program.

Input and output

However sophisticated the insides of computers may become, with their thin-films and their nanoseconds, a system is only of use if it can communicate with the outside world. It is in the area of input and output that most of our future problems will lie, since a significant error-rate is inevitable where moving parts are concerned. This must be catered for by not only detecting errors, but also, wherever possible, by allowing the system to carry on correctly in spite of them. This may involve automatic repetition of a faulty operation, or even the automatic amputation of a faulty device and its replacement by another. Such operations should be well within the capabilities of the supervisor programs of today.

Another function of the supervisor is the interleaving of the use that a variety of input and output devices makes of the central processing unit. Mechanical devices are necessarily slow by comparison with the central processor, and the scheduling of the use that autonomous input-output channels make of it is quite a complicated procedure, especially when the need to handle errors is taken into account. To cope with this situation it is desirable that all input and output operations are handled by the same set of routines, irrespective of the job being done, so again this becomes a function of the supervisor. An extension of this technique, sometimes called "multi-programming" or "time-sharing," allows several programs to reside in a computer simultaneously, each carrying on independent input-output operations while competing for control of the central processor. Some of the ways in which the hardware can help the supervisor in such operations will be mentioned in a later section.

The discussions so far have been suggesting that there is a need for both language processors (and other specialized programs) and for a supervisor capable of controlling all the component processors of the system, and of handling all input and output. These two together are commonly referred to as an "Operating System." This leads us to the next question, which concerns the overall size of any computer. The implementation of an operating system presupposes a certain minimum size of machine, since its functions are bound to require some significant computing power. It has long been accepted that the cost of a job goes down as the size of the computer goes up, provided the machine is fully-loaded. But whether it is desirable to use as large a machine as possible depends, from the user's point of view, on whether he gets

better value for money as a result. The penalty to the individual user for the use of a large machine in the past has been a decrease in its accessibility. With the advent of adequate operating systems this should no longer apply, since it is now possible to ask the machine to handle many jobs concurrently, and, for example, a short test can be allowed to interrupt a long job, have exclusive use of the machine, and then allow the interrupted job to continue as though nothing had happened when it has finished. Add to this the value to the user of having a machine capable of using powerful language processors even for small jobs, and it seems that the reasons for using as large a computer as possible are today stronger than ever.

Specialized application areas

In addition to the more conventional application areas of commercial and technical computing there are some specialized areas which require highly specialized systems to control them. One such area is that of data transmission with its problems of message assembly, error handling, on-line control from remote locations and so on. Another area is that of process control, with its problems of analogue-digital conversion, mathematical model-building, 24-hour service and the frightening possibilities of the immediate physical consequences of errors.

In his opening address to the Edinburgh Conference Sir Edward Playfair suggested that these might present another reason for not using one big computer (Playfair, 1964): "Are we sure that we shall in due course want to move things and to produce statistics from the same central processor? The requirements of complication and reliability are different." This argument has certainly been valid in the past, and may well still be so at this point in time, but in due course it ought to lose its validity if the computer is to be exploited to the full. As long as we regard a manufacturing process as quite separate from control of the stocks required in the process, which again are separate from the sales orders for the product, then we are justified in using separate techniques for processing each. But once we mechanize the data processing in these areas, or in the areas of accounting, payroll, work-scheduling, product design or any other such activities, we have a potential means of applying better management control to our organization. Computers are already being used as tools of management in all of these areas separately, but not many organizations have integrated their purchasing with their stock control or their payroll with their production recording, in spite of the obvious connections.

Once all the significant aspects of an organization's operations have been mechanized, there exists in the computer system a model of that organization, and, provided management know what questions to ask, and adequate information-retrieval techniques have been developed, then the computer can become a significant contributor to the overall control function of management. Further than this, one of the most important activities of management is prediction, and the very fact that a model of the significant aspects of the organization exists introduces the possibility of simulation, and makes economic tests of various management policies a real one.

Extensive activity in these areas is clearly some way off yet, but it does mean that ultimately it may well be desirable to perform many widely different types of operation on compatible computers. The problems of reliability and complication may well have been different for different classes of work in the past, but the differences are diminishing.

Probably by now any reader of this paper who is concerned with research, development or technical design problems is beginning to feel neglected. But this is not really so, as even his problems are at best part computation, part data processing. He will certainly express his problems in some language such as ALGOL or FORTRAN (let us not worry about trivial diversions as to which is better; neither is perfect, and most problems can be expressed adequately in either), so his computer will have to handle a great deal of language translation as well as technical computation. In any case, it is probably worth investigating how many designers working in industry produce what they think are new designs, when an adequate information storage and retrieval system would enable them to find a similar design produced three years before. Even in situations where designers have no illusions about their work being original, it is at present less trouble to start again than to search in hope of finding anything useful in history.

It appears that all potential computer users, irrespective of their class of problem, may well find themselves using what amounts basically to identical equipment, both in terms of hardware and of "software," as the operating system and its components are sometimes called. Obviously one of the reasons for this is to make it possible for manufacturers to produce enough computers to keep up with the demand. Another one, less immediately obvious, is in the area of real-time applications, where a second machine which is necessary for reliability need no longer be anything like the same size as the first, as long as the same programs can run on either. For this universal approach to be acceptable such computers must be extremely flexible at the hardware level in both size and power. However, as time goes on this burden of flexibility becomes more and more one of the functions of an operating system, and some of these functions will now be considered.

The functions of an operating system

The functions which an operating system must be able to handle to tackle all the areas of activity which have been discussed are as follows:

1. Automatic handling of a variety of processors. These must include standard language processors, other standard routines such as sorting programs and programs for setting up addressing structures for direct-access storage devices. In addition it must be possible for an

installation to include any processor of its own, and arrange for suitable additional control information to be recognized. The supervisor must be able to handle the transition between processors, and also carry out any accounting functions required in installation management.

2. Identification of input and output units. When programs are written in high-level languages it is desirable to avoid specifying the physical form of an input or output device within the program. The devices available in one installation may change; the number of data channels available may change; it may be required to run programs at other installations; library routines used in many installations must be easily adaptable to the equipment available. To cope with these situations, the supervisor must be able to establish correspondence, at the time a job is run, between the symbolic device names used in the program and the physical units actually available. It is desirable that these names should be independent of the types of devices available, so that a job may be run equally easily using for data files magnetic tapes, core storage, magnetic discs, or whatever else that may be available.

3. Control of input and output operations. To enable several autonomous input-output channels to operate concurrently the initiation and termination of such operations must be controlled by a program common to all jobs. This Input-Output Control System must be able to handle:

(*a*) error detection and error recovery procedures;
(*b*) data-transmission and process-control terminals;
(*c*) the sharing of the central processor by several input-output channels;
(*d*) the sharing of the central processor by several programs (when this can be justified).

The error control should be able to handle two classes of error: hardware conditions (with automatic error logging for the engineers' benefit), and programming conditions, such as incorrect labelling of files, or end-of-reel conditions on tape units.

4. Operating System Editing. An operating system which set out to be all things to all men would be so cumbersome that no one would use it. It is therefore essential that an editing facility be provided, so that the operating system may be modified to contain only those features needed by a particular user. Further, the editor program will be needed periodically as the requirements of an installation change, and as has already been said, it is a bad sign if an installation's requirements do not change for too long.

What price the operating system?

All these facilities cannot be provided at no cost, so it is as well to consider what their provision does cost. The programming effort the designer must make is inescapable, and should be just as much a part of his investment in product design as the calculations that go into his electronic circuits. It is no longer acceptable for him to regard the provision of software as a luxury to be added if possible at a later stage, and already the more shrewd purchasers of computer systems are looking at the software just as hard as the hardware.

The other factor in the cost of providing operating systems is the one most often complained of, the amount of high-speed storage they occupy. This complaint on the part of the user is as groundless as it is understandable. If only the designers would exclude the storage occupied by the system from the amount they offer the user, the problem would not arise. Obviously the amount of storage occupied by the system will vary, but the amount offered should be reduced at least by the minimum the operating system may require.

How the hardware can help

If operating systems are to be implemented effectively, their functions must be appreciated at the time when the hardware is designed. Features necessary to make an operating system easy to implement are these:

1. Automatic interruption of the central processor. The hardware must be able to cause an interruption of the program currently being executed, irrespective of what is being done, and without the knowledge of the writer of the interrupted program. The events which require this are, for example, the completion of an input or output operation, the discovery of an error, or a request from a remote terminal for servicing. Such interruption facilities bring with them many detailed problems which must be considered, such as the method of determining which interruption should have priority if two are waiting to be processed.
2. Storage protection. It must be possible to prevent a programmer from interfering with parts of the machine which are not concerned with his program, and especially with parts where the operating system resides. It must be possible for the system itself to override these restrictions, while any attempt to do so by a user should cause an interruption as described in item 1. It is helpful if it produces a message saying what he has done wrong, before returning control to the operating system to terminate the job. The programmer should also be able to specify his own protected areas.
3. Error detection. Errors of all sorts, whether invalid data input, machine errors or program errors must be detected reliably, and correction or by-pass action undertaken, so that the running of the system is not interrupted unnecessarily. If the machine can assist the maintenance engineer by telling him where it feels poorly, so much the better. A great deal can now be done in this direction with built-in hardware diagnostic programs, which automatically check the correct functioning of the various components of a system. It is an interesting paradox that the importance of error-checking increases as

the realiability of equipment increases. Whereas there was a time when no one really expected computers to give the right answers all the time, we now have a right to expect that, if we get an answer at all, it can be relied upon.

4. Relocation. Languages which take no account of the machine configuration, and which also allow pieces of program which have been written and translated on separate occasions to be run together, make necessary an efficient means of relocation. This is particularly important for programs which exceed the size of the available machine and have to be run in several phases. Sections of coding may be executed from any part of the store, and the programmer cannot (and should not) know where. So the operating system—or more particularly the loader—must be able to load different sections of a program into whatever locations are required. This function has until recently been performed entirely by programming in the loader program—but it is an operation in which the whole system can be made much more efficient if the hardware and software designers work together.
5. For a computer to be universally acceptable, its machine-language instruction codes must be common throughout a complete range of size of computer. It is no longer good enough to expect the user to embark on a marathon re-programming operation simply because his business has expanded enough to require a larger computer. Compatibility at the level of high-level languages is a good start, but compatibility at machine-language level is even better. Clearly it will be worthwhile to provide some optional features on some versions of a machine: exclusively commercial users who are very pleased with their powerful editing instructions do not take kindly to having to subsidize standard floating-point hardware for the benefit of their technical friends. But now that the engineering techniques available make it possible to modify the computer instruction set very easily (by producing an economic form of read-only storage system similar in concept to the wired store in EDSAC 2 (Wilkes, Renwick and Wheeler, 1957) or the Manchester University slug store (Kilburn and Grimsdale, 1960), the basic framework of a central processing unit can be common to all application areas. Another point not to be overlooked in this area is historical compatibility. The designer cannot penalize all his existing customers (who have probably provided a lot of his resources to develop new computers!) by developing machines on which existing programs will not run. So compatibility between the past and the future is equally important.
6. Standard interface for input and output. It appears that coming generations of central processing units can still vary, but that is nothing to the variations we can expect in input-output devices. We already have to cater for everything from a Telex line at a few characters/sec. to a radio link at hundreds of thousands. To do this, to make allowance for visual and audio devices which are the latest arrivals, and at the same time to allow for future devices which have not yet been thought of, the only possible solution is to define a "standard-interface." As long as all devices conform to a formal set of conditions, the input-output channels on a computer can handle whatever devices may be connected.

Many computers have, of recent years, had several of the features just listed, but not until the recent announcements of compatible computer ranges like the IBM System/360 or the I.C.T. 1900 have systems become available which should have all these features simultaneously. This does not mean, of course, that the end of the trail has been reached, either for hardware or for software. It is, rather, the beginning, since it should now be possible to incorporate improvements in hardware without the user even being aware of it.

It is not as easy to make changes in programming languages without the user's knowledge as it is in hardware, and although he may be excused for not welcoming changes, he must accept the fact that computer languages are living languages just like spoken languages. They must therefore develop to take account of changing circumstances. Being defined more formally, we have more control over when changes should take place, but changes there must be. Once this is accepted, then the use of an operating system does a great deal to simplify such problems for the user.

The operating system and the user

Many of the operations which used to be regarded as the proper sphere of the program, or more accurately, the programmer, are, with machines at the stage of development now reached, becoming functions of the hardware. It was suggested recently that computers may reach the stage one day where "efficient equipment 'is grown' to replace those portions of the stored programs which have been proved successful or optimized in some sense" (Carr, 1964). This may well take some time, but already some programmers fear that the removal of some of their responsibilities is reducing their importance. This fear is not justified if the programmer faces his real job, which is to solve problems. He should be grateful for any developments which allow him more freedom to devote his time to improving his techniques in the area he should be studying, namely problem solving.

Today's programmer must certainly accept the fact that he no longer understands the purpose of many instructions which appear in a program he has written. But he has long since given up expecting to understand how the hardware of his computer functions, so why should he worry about the software, especially as the line between them is becoming so flexible? There will always be a

need for skilled professional programmers to develop operating systems and to push the available equipment to the extreme limit of its capabilities, but their job is no longer the same as that of the programmer. If a programmer (or a man-with-a-problem, as he should perhaps be called) does feel the need to descend from his high-level language to an assembly language he will find in that language many features which are of very little use to him. They are only there for the benefit of compiler-writers and other professional programmers, and rightly so, since they should be the most frequent users of such levels of language. No argument is really as simple as this, and there may well be situations in which a problem-programmer needs detailed knowledge of his computer to avoid making inefficient use of it. This will still matter until computers are given away free in exchange for the wrappers from six input-output devices, and that will not be for a little while yet. The chances are that the best systems-programmers will be discontented problem-programmers, since they will be in the very best position to appreciate the users' real needs.

As a direct result of the use of operating systems it is possible economically to turn the man-with-a-problem into a problem-programmer able to communicate easily with an extremely powerful problem-solving tool. It may appear at the moment that he is being kept further away from his new tool by the operating system than his predecessor was. This may be true, but he is nevertheless as a user already getting better service from it, and all the hardware and software are now available to make it appear that each user has exclusive use of the system. It was a most fitting conclusion to the Edinburgh Conference that Dr. Wilkes should have shown this so clearly in his demonstration of the use of the MIT Compatible System. It is to be hoped that it will not take too long for the economic situation to allow us to put man's time at a higher premium than that of a machine.

References

BARRON, D. W., and HARTLEY, D. F. (1964). "The Influence of Automatic Programming on Machine Design", (*UKAC report*, p. 21).

CARR, J. W. III (1964). "The Future of Programming and Programmers," *The Computer Bulletin*, Vol. 8, No. 1, p. 9.

KILBURN, T., and GRIMSDALE, R. L. (1960). "A digital computer store with very short read time," *Proc. I.E.E.*, Vol. 107B, p.567.

PLAYFAIR, Sir EDWARD (1964). "Computers and psychology," *The Computer Journal*, Vol. 7, p. 1.

UKAC report on 1964 Edinburgh Conference: "The Impact of Users' Needs on the Design of Data Processing Systems," Organized by the B.C.S., Brit.I.R.E. and I.E.E.

WEGNER, P.—Editor (1964). *Introduction to System Programming*. Academic Press, London.

WILKES, M. V., RENWICK, W., and WHEELER, D. J. (1957). "The design of the control unit of an electronic digital computer," *Proc. I.E.E.*, Vol. 105, p. 121.

Book Review

Management Standards for Data Processing, by DICK H. BRANDON, 1963; 404 pages. (London: *D. Van Nostrand Co. Ltd.*, 93s.)

In the past few years, the literature of Data Processing has grown almost to a flood. Most of it has poured over the specialist—analyst, programmer, engineer—dealing with technical matters pertinent to his field. Much of the rest has been directed at managements, explaining the intricacies of the subject, and quieting their fears of its impact.

There has, however, been little of direct service to the harassed D.P. executive whose daily lot it is to turn the eccentricies of his wayward (if brilliant) brood into the targets achieved and work accomplished that alone can justify the existence of a computer installation. Mr. Brandon's book will do much to ease his sufferings. Clearly the content has been distilled from experience, and will prove to be an invaluable guide past the pits and traps which lie in wait for even the most experienced of those who seek to set up an efficient, productive, controlled data processing organization.

The theme of the book is Control, and its development is concerned with the establishing of standards to which every aspect of D.P. work can be subjected. To this end, all the software components (and a little of the hardware) are meticulously dissected, classified and coded, and their relationships with one another carefully defined. From the skeleton thus exposed Mr. Brandon builds up a series of suggested techniques by which the D.P. group can be started on the right path, and kept from wandering too far from it.

Systems Analysis, Programming, Operations Performance Evaluation are the major topics with which the book deals, and each is stuffed with formulae and numerical information. It would be unwise to apply the former to any specific case (other than the originating one) as they stand, but they do serve as a most useful starting point.

The essence of control in most organizations lies in adequate (but minimal) documentation, and it is heartening to read Mr. Brandon's constant iterations on this matter.

The book is specially recommended for systems analysts and programmers. They would then look with less disfavour upon the efforts of their superiors to guide their creative talents into narrower channels.

C. E. HARDING