Operating Systems:

their Purpose, Objectives, Functions, and Scope.

C.A.R. Hoare.

Summary: This paper proposes a definition of the purpose and objectives of an Operating System, which is intended to clearly distinguish the responsibility of its designers from that of the designers of hardware, programming languages, utility routines, library procedures, and other software. The definition is then applied to delimit the proper functions of various Operating Systems, working to meet various requirements in various different circumstances.

This paper has been submitted to the first International Seminar on Operating System Techniques held at Belfast, Northern Ireland, August 30 to September 4, 1971.

1. Purpose and Objectives.

The basic purpose of a computer operating system is to share the hardware which it controls among a number of users, making unpredictable demands upon its resources; and its objectives are to do so efficiently, reliably, and unobtrusively.

1.1 Amplification.

The definition of a user is left deliberately vague; it may be a single person or a team; a single program or a suite of related programs carrying out some task. Each user is regarded at least on occasion as more or less isolated from all other users, and is not supposed to be concerned or aware of what the other users are doing at any given time. Consequently an operating system should protect each user from the accidental or deliberate effects of the actions of other users; this is accomplished by presenting to each user an apparent or virtual machine configuration (usually smaller and/or slower than the actual hardware available), which that user may regard as his own separate machine, relatively independent of the virtual machines presented to other users.

The main problems faced by an operating system arise from the unpredictability of the demands made by its users. If all demands were made in predictable combinations at predictable times, the whole operation of the machine could be planned in advance by a single prewritten program. Indeed, any ALGOL program using block structure (or FORTRAN program using EQUIVALENCE for a similar purpose) is "sharing" the store of a computer among different "users" (blocks or procedures) at different times. But since the pattern of sharing is completely predetermined by the program, one cannot argue that such a program is an operating system.

The efficiency of the sharing achieved by an operating system can be measured in terms of the density of utilisation of resources. Each item of hardware (e.g. peripheral device, word in core store, channel, central processor) is allocated to a user (on his request) for only a certain percentage of the time. Each such item can be given a comparative weighting related to its capital and running cost and possibly also scarcity value; multiplying each weight by the percentage of time in use, it is possible to obtain a figure for the density of utilisation of the installation as a whole.

Naturally, this figure will depend on the degree of balance between the hardware configuration and the user workload; but even in the case of perfect balance and a well-designed operating system, unpredictable variations in the behaviour of the users, and their demands for good service, are likely to place quite a low limit on the efficiency achievable.

The reliability achieved by an operating system must obviously depend on the reliability of the underlying hardware; nevertheless, a good operating system should certainly mitigate the effects of occasional hardware malfunction, for example by automatically repeating unsuccessful operations, or restoring corrupt data; and if automatic recovery is not possible, an operating system should confine the effects of error to as few users as possible, and assist each user in making good after a breakdown. To some extent, the objective of increasing the apparent reliability of the hardware also derives from the primary purpose of sharing a computer; for the effects of hardware failure on a shared machine would be many times as serious as on a single unshared computer, in that it will affect many more users; and also, on a shared computer, no single user without software assistance would have sufficient understanding or control over the installation as a whole to enable him to take steps to recover from a fault in the sort of ad hoc way which is often possible on a single stand-alone computer.

Another objective of an operating system is to give each user the same predictability of service that he would have on his own separate (slower) computer, to which he would have immediate access whenever he wished. A user should know the apparent speed and capacity of his virtual machine, and thus be able to predict how long he will have to wait after submitting his program and data before getting his results back. If he is disappointed in his expectation (which he is sure sometimes to be), it does not matter whether this is due to a hardware breakdown, or to a deficiency in the scheduling strategy of the operating system. Thus the achievement of reasonable predictability of turnround for the individual user is an important corollary of the objective of reliability; but it is difficult to achieve in view of the unpredictability of the workload presented at any given time by the other users of the system, and there is no doubt that a user must occasionally be disappointed in his predictions.

There is one clear consequence of the view that the purpose of an operating system is to share a computer installation among a number of users: for the individual user, the use of the operating system is not optional but

compulsory. He must therefore regard an operating system as a necessary evil, tolerated as the only means of obtaining a share of an expensive computer installation, when he would really prefer to have exclusive use of a rather cheaper one, which he presumably could not afford. He is in the position of having to travel by bus, because he cannot afford the private car which he would prefer. Thus as far as the user is concerned, the best operating system is one which is most unobtrusive - which appears least to stand between his requirements and the virtual-machine which is to satisfy them. The complex control-languages and obscure output messages which are such a feature of modern software are certainly widely recognised as evils, even if it is claimed that they are necessary.

There is another consequence of the compulsory nature of the use of an operating system, which should be taken to heart by its designers and implementors: and that is the obligation upon them to achieve very highest quality of their product. Any other item of software is at least in principle optional for the user; if the quality of a library program is unacceptably low, he can write or select a better program to use instead. But in the case of an operating system the individual user has no choice; he cannot replace the parts which are of substandard quality. It is to be hoped that future designers of operating systems will accept the obligation of high quality; will recognise that quality is measured by efficiency, reliability, and unobtrusiveness; and will produce software which is a major improvement on current products in widespread use, which display none of these qualities in any high degree.

1.2 Alternative views.

It must be recognised that the unsatisfactory nature of many current operating systems is not wholly due to lack of good will or competence on the part of designers and implementors; but rather to the fact that they started with a view different to that proposed here both of the purpose of an operating system, and of its objectives. These alternative views will now be discussed.

(1) The Universalist view.

One widely accepted view is that the term "Operating System" should cover the entire range of software support offered by a computer manufacturer. Naturally, there is no point in arguing about terminology; nevertheless, indiscriminate use of a single term to cover so wide a range of software

is very unhelpful, and may have serious consequences in blurring the functions and objectives of different items of software. In this paper we choose to adopt a more precise terminology, to single out some important particular part of a manufacturer's software range.

(2) The Compatibilist view.

A second view, which certainly underlies the practice of many manufacturers' operating systems, is that the objective of an operating system is to secure "compatibility" for user programs so that they can be transferred between installations of widely differing speed, size, and configuration. Naturally, it is a sensible objective of a manufacturer's commercial policy that the lower machines of a range should be identical in their hardware structure to the virtual machines which are set up by an operating system within the more powerful shared installations. Furthermore, there is an obvious role for simulators, emulators, service routines and library programs, and machine-independent languages, which will assist in the transfer of programs from one machine to another. But it should not be the province of an operating system to enforce compatibility when this is not wanted or appropriate for the user; for compatibility is seldom achieved without a cost which surprises user and implementor alike.

(3) The Perfectionist View.

Finally, there is a widespread view that an operating system should present to each user a virtual machine which is in some way preferable to the original hardware, in the sense that it possesses more facilities or is easier to program. This view arrogates to operating system designers a number of responsibilities which are better discharged by others: for example the provision of better hardware should be the responsibility of hardware designers; useful facilities should be provided by designers of service programs and library routines; and the task of making a computer easier to program belongs to programming language designers and implementors. In the past, when presented with badly designed hardware and even worse programming languages (which cannot be changed for reasons of compatibility), there has been a great temptation for the operating system designer to attempt to remedy the situation by software. The resulting systems have inevitably grown very large and expensive, and so far from mitigating previously existing defects, they have introduced many new ones of their own. There are few writers of major programs who would not prefer to

interface directly to the hardware rather than to an operating system.

The main reason for taking a very strict view of the proper province of an operating system is that the use of an operating system is compulsory on a shared installation. Any resources of core storage, backing storage, channel capacity and processor time which are used by the operating system present a permanent and inescapable overhead on the installation and the user; whereas any compiler, library routine or service program which is invoked by a user may be selected, adapted or even specially written for his particular requirements. Even worse, any attempt to present a user with a "more convenient" virtual machine than that provided by hardware must almost inevitably be based on assumptions about the nature of his use of the machine, which will in particular cases be unjustified. Thus the provision of additional software support as part of an operating system nearly always leads to loss of flexibility in application. A machine empty of software is always the most flexible; the addition of even a few instructions of "compulsory" software can only reduce the range of its application, never increase it.

Nevertheless, as in all successful design, a certain scope for compromise must be recognised, and the following special cases can be made:

(i) When the operating system itself needs a service, (for example binary-decimal conversions) such a service can often be made available to users at no extra overhead.

(ii) When considerations of compatibility and convenience are very strong, and the extra overhead quite small, a suitable concession can be made.

(iii) The hardware designers must be permitted to interpret instructions by software if on certain machines in a range this is more economic than building their interpretation into hardware.

As mentioned above it is a legitimate objective of an operating system to simulate a virtual machine which is more reliable than the actual hardware of the machine, and software services associated with this objective may legitimately be ascribed to an operating system.

2. Function and Scope.

This section surveys the range of functions which an operating system is properly called upon to perform in sharing a computer installation among many users. The survey starts from the simple rudimentary forms of operating system suitable for small machines and specialised applications, and progresses to the larger and more complex systems providing a service to many users with differing needs.

2.1 Rudimentary systems.

It is instructive to start a survey with an "empty" or "null" case where no operating system is needed at all. Such cases are found in the application of computers to laboratory automation or process control, where a small computer is wholly and continuously dedicated to a single task. In an application such as airline seat reservation a larger computer is generally used, and it shares its attention among a number of separately identifiable tasks or transactions. But usually the paramount importance of reliability and predictable response times in these applications force the programmer to work out fairly precisely in advance the potential "behaviour" of the transactions, and to take advantage of this knowledge throughout the design of his sharing strategy and its implementation. To date, it has proved impossible to design a "real time" operating system which will give real-time service on a general bureau basis. We shall therefore omit real-time applications in the remainder of this survey.

Returning to a more conventional environment, where users submit more or less disjoint jobs to be run on a computer, it is obvious that the simplest method of "sharing" is just by succession in time. Each user follows another, and has sole use of the machine during the period allotted to him (usually controlled by a booking system). The role of operating system software is minimal: its main function is to reset the machine to some known state at the beginning of each user's session; and if some suitable output device is available, it may provide the facility of dumping the user's job if his period expires before the job is complete; and reinstating it at the beginning of the next session. The facility for reinstating programs may be used to load service programs or compilers on behalf of the user. This operating regime is suitable for small machines installed in scientific laboratories.

In a commercial environment a similar method is quite suitable for a small machine. But such an installation should not just be regarded as only a central processor, but also as the set of removable tapes, discs, etc., which contain essential user files; and which are usually stored centrally in or near the computer room. Thus when each user starts his session, the files relevant to his job must be set up by operators, and there is a severe risk that the incorrect data will be mounted. Thus the user may inadvertently process the wrong data; or worse still, overwrite valuable information still required by another user.

The solution to this problem is to write a header label at the beginning of each file, containing the name of the file and the date up to which the information must be preserved; and these are checked before the file is read or overwritten. This obviates a reliability problem arising out of the sharing of the computer, and is thus properly an operating system function.

Another function connected with reliable sharing is the error correction and recovery procedure carried out by standard read/write routines. But perhaps these should rather be regarded as part of the "hardware", in that it is a choice of the hardware designer how to minimise the cost of reliability by a mixture of hardware and software techniques.

2.2 Batch Monitors.

As machines get faster, the time taken to execute a typical single job submitted by a user reduces until it is very much less than the time taken by the manual changeover between one job and the next. Indeed in a scientific workload with a high component of program testing, a significant proportion of jobs will fail before their first few seconds. The reduction of the interjob gap may be achieved by introducing a batch monitor to automate the transition between one job and the next, and to replace intervention by the user or operator. The main additional function of a batch monitor is to detect the end of a job either by expiry of a time limit or exhaustion of input data, the overflow of some resource constraint, or other detectable error; and to indicate to the operator the point where the output of one job should be separated from that of the next.

After virtual elimination of the inter-job gap, the main limitation to the throughput of the machine is the slow speed of the input and output devices (card readers, line printers, etc.); it is essential to minimise the time

spent waiting for these devices, particularly during periods of card loading, paper change, routine maintenance, and hardware breakdown. This can be achieved by off-lining of information: jobs and data are read from a slow device onto backing store as a continuous operation well in advance of processing; and information destined for printing is also written first to backing store, and later transcribed to a slower device as a continuous operation. When off-lining is carried out simultaneously with computation on the same computer, it is known as pseudo-offlining. The continuity of the input/output ensures that the slower devices operate at full speed for as long as they are in service and there is work for them; and when they are out of service, the computer can continue transferring information to and from the backing store until they are operational again. Thus an improvement in efficiency and reliability is simultaneously achieved.

When the information is transferred between main store and backing store, there is usually a large average overhead on each transfer; it therefore pays to perform input and output in fairly large blocks, each containing some ten to fifty cards or lines of information. Thus information from cards is assembled into blocks before being written to backing store, and in reading back, the blocks are disassembled into lines again before being presented to the user program. Similarly, lines destined for output on the printer are assembled into blocks before being written to backing store, and disassembled again on printing.

If the hardware of the computer permits backing store transfers simultaneous with computing, the operating system can further increase efficiency by a buffering scheme, which allows the user program to proceed while a block is being output, and attempts to keep one block ahead on input. This blocking, unblocking and buffering is a function of the operating system, since it improves the efficiency of a shared installation; and it should be of no concern to the individual user whether his input and output information resides for a period on backing store or not. The part of the operating system which administers the above-mentioned aspects of an operating system is known as an input/output control system.

When pseudo-offlining is used, and the central processor develops a fault which is repaired, it is very important to be able to continue output of the results of jobs which had been completed before the fault occurred; and to continue processing jobs which had been previously input, preferably starting again the job during which the fault occurred. Such an action is known as a warm start, and is one of the means by which an operating system limits the potentially more extensive effects of hardware unreliability on the efficiency and service of a shared machine.

One major disadvantage of a batch monitor is that once a long job has been initiated, any short jobs input later must wait until that long job is complete, this leads to long and unpredictable delays in the turnround of the majority of jobs, which are short. The inconvenience can be mitigated if the operating system has the power to select for execution an arbitrary job from the input queue, and thus give favour to shorter jobs even if they were input after a longer one. An even better service can be given if a long job can be interrupted and dumped to backing store to permit short jobs to pass by. This function of an operating system is known as job scheduling.

In a batch monitor system, programs which require operator action, for example, the mounting and dismounting of tapes or discs, present particular problems, since such a program will in general have to wait completion of operator action before proceeding. This problem can be mitigated by the job scheduler, which can ensure that the proper files are mounted before selecting the job for initiation. Nevertheless, the mounting of special files inhibits "queue-jumping" by shorter jobs; and furthermore if a high proportion of jobs requires such action, the efficiency of the computer installation will be severely limited.

This problem can be solved by keeping the information required by most users on a large disc store, from which it is immediately available on demand. The task of sharing a disc store among many users falls to the filing system which splits the disc space into areas, and allocates and deallocates them to each user on request. Thus each user of the filing system has a virtual machine with a "non-volatile" backing store, which can store information between one run and the next. In fact one of the major problems

of a filing system is to guarantee absolute permanence of stored information, in face of disc hardware failure; or even worse, breakdown of the central processor during the process of updating vital information on disc. The required security seems to be achievable only by periodic dumping of information from disc onto magnetic tapes, which can then be removed from risk by dismounting. A second requirement of a filing system is of course to ensure that no user can deliberately interfere with the information belonging to another user, either by reading or by writing. The third requirement is to provide an environment in which a library of commonly used programs and subroutines can be stored on disc, and efficiently shared by all users. Finally, a filing system may also provide a communication facility which permits one user to give to selected colleagues a controlled access to his files.

A further function which falls to an operating system is that of maintaining a log of significant events. The purpose of this is threefold:

(1) To assist in diagnosis and recovery from error or breakdown.

(2) To provide statistical information on the workload and the performance of the operating system, so that it can be improved.

(3) To provide the information for charging each user for the resources he has used. The calculation of the charge and the billing of users may also be regarded as an operating system function.

2.3 Multiprogramming Systems.

Up to this point, our survey has assumed that only one user program is being executed at a time. However, on larger machines, a denser utilisation of resources may be achieved if several programs are initiated together to run simultaneously, in a mode of operation known as multiprogramming. If there is more than one central processor in the system, multiprogramming is needed to make effective use of the extra computing power. Even if there is only a single processor, it may pay for it to alternate attention between several programs, and thus avoid delays while one program is communicating with backing store or with the operator. The part of the operating system responsible for sharing a central processor or processors between several concurrent programs is known as the dispatcher.

When a general facility of this kind has been set up, it is convenient to allow the operating system itself to take advantage of it,

and to delegate certain of its tasks (for example, pseudo-offlining) to programs which run in parallel with each other and with the user's programs. But operating system tasks differ from normal user programs in that they require to communicate with each other while they are running; and each communication may involve some form of synchronisation. Similarly, user programs must occasionally synchronise with each other if they are competing for some limited resource. Some means of administration of synchronisation must be part of the responsibility of the dispatcher.

In the absence of multiprogramming, it is assumed that the user program, once initiated, will have access to all resources of the computer installation other than those occupied by the system itself. In many cases, a proportion of these resources will not be required at all for a given job; and in other cases, the resources are required only for a proportion of the total running time of the job. One of the main advantages of multiprogramming on a large machine is that the unused resources can be allocated to another program. Thus the task of resource allocation belongs to the operating system. The objective of resource allocation is to give each user the resources he needs exactly when he needs them, and thus secure a general high density of utilisation. In some systems, the user defines the required resources for his job, or each job step, by means of job control cards included in his submitted job; the interpretation of these is the task of a resource allocator. Of special interest is the management of core storage, which is usually one of the most valuable and scarcest resources in a multiprogrammed computer installation; many ingenious schemes (paging, segmentation, etc.) have been devised to postpone allocation of core storage to programs until the very moment that they first need it, and to take areas of core storage away from programs which are not currently using them and to lend them to other programs; and also to share subroutines and data among programs which may happen to be using them simultaneously.

With the advent of multiprogramming, it again becomes economically feasible to permit the user to communicate with his own program while it is running, in exactly the same way as he would if he had a machine of his own. The provision of a console on the user's virtual machine involves no new principle in the functioning of an operating system; nevertheless, the strict requirements for good response time and acceptable cost entail that many of the techniques used to carry out the major functions of an operating system must be specially designed for this new mode of operation.

3. Conclusion.

This paper has put forward a theory of the purpose and objectives of an operating system, and has shown how many of the traditional functions of operating systems arise from their primary purpose of sharing a computer installation.

The paper contains no new technical proposals to solve the many problems which face implementors and users of operating systems at the present time. Nevertheless, it is hoped that the general philosophy propounded here may prove to be of some general benefit:

(1) To the users of an operating system, it explains the major source of the problems, namely the provision of a predictable service in the face of a highly varied and unpredictable workload; and suggests that disciplines designed to reduce variability and unpredictability may show a valuable return in the form of improved service and reduced cost.

(2) To the designer and implementor a clear statement of purpose and objectives of the system as a whole and each part of it may be an invaluable guide in taking the many thousands of decisions of principle and detail involved in any large programming system.