DATA MANAGEMENT AND I/O

Mr. W. A. Clark

Department D88, Building 931,
IBM Corporation,
Poughkeepsie,
New York, 12602.

Abstract:

The various problems encountered in the design of general data management and I/O systems are surveyed and tools which are useful in solving the problems are discussed. An approach to teaching this material is presented which emphasises consideration of alternative solutions.

Rapporteurs:

Dr. J. Eve
Mr. P. Henderson

1. Course Prerequisites

Mr. Clark indicated that a course in data management systems would require among its prerequisites:

1. Communication Theory, in order to provide an intuitive grasp of concepts related to system balance;
2. Graph Theory, to assist in formulating and clarifying thoughts;
3. Queueing Theory, which can be applied to both communications aspects and efficient use of storage within the data management system (familiarity with both simulation and analytic techniques is important);

and 4. A programming language which is convenient for formulating combinatorial problems (APL is a good example), since, in considering path finding algorithms and problems in networks, a combined statistical and deterministic approach is often needed.

2. Course Outline

A course description was suggested, formulated as a list of general topics, some of which are treated in more detail below; unfortunately, there is no textbook available at present which covers this material.

2.1 Structure of Data Management Systems

Presenting a historical view of how data management systems had been constructed was, in the speaker's view, not a good approach, it being better to concentrate upon and emphasise the alternatives available in a data management system.

2.1.1 A data management system may deal directly with I/O devices, or manage them indirectly through a storage management system which deals exclusively with the handling of data flow among devices (perhaps even including main memory). The complexity of current systems is a strong argument in favour of the indirect approach. At the very least the alternative of separating storage management and data management should be made apparent.

2.1.2 The concept of implicit I/O, embodied in virtual memory systems, should be contrasted with the more usual concept of explicit I/O as seen in 'GET' and 'PUT' type statements. Virtual memory, which is usually viewed as a means of providing a large main memory space for the user's convenience in organising his data, necessarily contains much of the management formerly handled explicitly by GET and PUT statements, and from this viewpoint, it is not difficult to imagine a

completely hierarchical storage system extending right out to the shelf which is entirely implicitly managed. In discussion of this topic, the distinction between linear address space and segmented spaces should be emphasised.

2.1.3 Data management systems are concerned with binding of various kinds. The opportunities to effect binding should be emphasised and contrasted, and, while potentially the number of items to be bound is very large, the problems may usefully be discussed in terms of four classes of item, namely devices, data, access language and code.

Data management systems bind devices to programs. Formerly this was done at compile time, but now can be done at the time at which a job is scheduled, or indeed more dynamically. In general, the more dynamically the devices are assigned, the more useful the system and thus a clear statement of the binding choices available is important.

Usually it makes little sense to consider a time other than compile time in choice of an access method; however, at times, the choice of specific functions used in managing a particular collection of data may be arranged in a more dynamic fashion than a pre-specified choice at 'open' time. The choice of subroutines to effect transfer of information between elements of the data base can be made as early as compile time or postponed until a macro or command is interpreted.

2.1.4 In designing a data management system an interesting choice in the construction of directories is the decision whether to orient them to data or to storage. Typically, directories are regarded as directories to data, in that a directory is generally entered with a symbol related to the data required and the whereabouts (volume, track, etc.) of the required data is obtained from the directory. Other organisations which will occur more frequently in storage management or implicit I/O systems will be storage oriented, in that the directories will display the contents of a memory device. Consider the eight associative registers of the 360/67 which are used to reduce the number of page and segment table accesses: these form a storage oriented directory and may be contrasted with the page tables which are data oriented. If one attempts to build systems which are entirely hierarchical, including two or three levels of memory (core, film and monolithic) as well as such storage devices as drums and discs, it is probably more appropriate to consider storage oriented directories at all levels except the most extensive.

2.2 Tools needed in information systems

2.2.1 Some treatment of inverted files and the motivation for these must be given. The speaker believed that, in the future, motivation would lie not so much in the need for efficient random access to arbitrarily distributed data, as in the consequences of data being shared by many users, it thus becoming increasingly important to obtain unique representations of data permitting a datum, which is changed by one user, to be immediately available to other users. It is convenient, therefore, that such a datum should appear only once in the data base. Logical structures which define records and files will thus have to become truly structural, as opposed to the multiple copies of information currently in use to represent different logical structures.

2.2.2 Redundancy and recovery systems will be increasingly important, and the speaker stressed the necessity to go far beyond ensuring the existence of a 'back-up' copy of the previous generation of information. For audit trails, ideally, a time ordered journal of changes to the data base is desirable; however, the cost of current memory devices precludes this. In the future holographic devices may achieve costs as low as 10^{-6} cents/bit - one tenth of the cost of paper - such a low cost virtually eliminating the purge problem. However, solving the indexing problem to such vast amounts of information will be of paramount importance.

2.2.3 The choice of an appropriate indexing technique, from the number of techniques known and under investigation, must be understood. For example, the IBM 2311 and 2314 discs permit an associative hardware search involving the reading of identifiers of successive records as they pass the reading heads and transferring that record with an identifier matching a key. However, the use of this apparently powerful feature, to locate portions of the index itself, has the effect of utilizing the main memory, the channel, the controller and the device for, on average, half a disc revolution per record read. A better scheme would clearly be a straightforward blocking of the index entries, thereby freeing these components during the 'latency' period.

2.2.4 The speaker also noted that an understanding of the accessibility of remote data was required. For example, large information systems will contain the whole warehousing problem with its associated problems of minimising inventory costs, transport costs, etc. In addition, large, dispersed firms will have requirements for dispersed data, but with the data used most often locally available; however, since information will tend to be uniquely represented, it must also be available

to remote parts of the system, thus requiring decisions as to which information should be remote, which should be local and indeed how to address any of it. Mr. Clark suggested that this was an area of study in its own right.

2.2.5 An important and usually forgotten topic in the development of information systems is that of conversion. It was suggested that the two main types of conversion problem which should be considered were:

1. In going from batch to information systems, the need to obtain valid data in a representation which is suitable for random retrieval;

and 2. Conversion of the type needed to go from one generation of a computing system to another.

Probably the most important tool for conversion is a good system of descriptors, as the handling of descriptive systems requires that another level of description is available. Mr. Clark pointed out that much time has been spent recently devising standards for disc and tape labels and suggested that some of the items, included in these labels, should not be so rigorously treated as they were part of a more extensive system which so far has not been specified. It therefore seemed likely that these standards would be entirely inappropriate to the real needs. If this assertion is true, it follows that the only standardization tenable at present is that of a representation of the description of descriptors!

2.2.6 The problems of interlocks, although they are not specifically problems of data management, are important in these systems and some understanding of them is a necessary background to data management. In order to maintain the integrity of data, interlocks must be provided by the system and the effect of these interlocks on the rest of the system can be profound. A number of papers on the problems of deadlock have been published recently by the ACM, mostly in relation to device assignment. More interesting in the present context, in addition to the logical problems of preventing deadlock, are the problems of information interlocking; reducing the contention that arises when an interlock is set, immediately reduces the order of the multiprogramming needed to achieve a given volume of throughput.

2.3 Communications

The teaching of communications, both digital and analogue, has typically been left to departments of electrical engineering rather than computer science departments, and the speaker suggested that this might be changed with advantage.

2.3.1 In his view, the foremost topic was that of dealing with terminals, where one must give attention to:

1. the proper use of terminals from the human viewpoint;
2. effective use of the line facilities available;
and 3. the problems of attempting to maintain compatibility among a variety of terminals.

It was pointed out that, while 3. is almost impossible in general, something could be achieved with the key entry terminals (including simple CRT terminals), by a classification of their methods of control, buffering schemes, polling methods and line control schemes.

2.3.2 Some discussion is needed on available tariffs, line concentration and the various techniques one can use to reduce the cost of communication. The most difficult parts of I/O systems to construct are those concerned with communication, which are not really part of a data management system, but have come to be implemented together simply because they interface with control hardware which is itself shared by the data management system. It was suggested that very few people understood the control of all kinds of terminal sufficiently well to be able to optimize it.

2.3.3 Mr. Clark indicated that the use of unreliable asynchronous channels is not understood at all by the average system designer, and that the practical application of results in communication theory is becoming increasingly important. Communication networks between computers have introduced the need for high bandwidth channels, and the equipment currently in use tends to exploit multi-level codes. However, since error rates are very high, it is necessary to be able to handle these systematically.

2.3.4 Finally, Mr. Clark mentioned the topic of communication security and noted that very little of the work done in this area has been discussed in the context of computer science.

2.4 Subsystems or Central Control

Mr. Clark suggested that one of the major problems in the design of data management systems lay in deciding whether the control of a set of tasks should be delegated to a subsystem or retained by the main system. He took, as an example, the regeneration of information on a graphic display terminal and pointed out that this could take place from main memory, from the memory of a

private computer or from a controller which does not execute instructions but simply caters for regeneration. Architecturally there is considerable freedom of choice, (e.g. in the IBM 360/25 there is apparently a channel, a controller and a device; the actual implementation uses only the CPU and a device), and the problem of selection is important. However, the advantages and disadvantages of subsystem versus central control are not well understood.

2.4.1 There are two primary reasons for partitioning a system into subsystems:

1. the increased responsiveness one obtains from a subsystem, (it is difficult to imagine a large scale system such as an IBM 360/75 tracking 100 light pens);

and 2. the boundedness arising from building a subsystem which manifests itself in many ways of which perhaps the reduction in context is most important.

The speaker considered the example of a system in which a processor is dedicated to a single graphic terminal such that, if an interrupt occurred it could only have arisen for one of very few reasons and for which the required action is fairly obvious, it not even being necessary to save the entire state of the processor. He contrasted this situation with the complex interruption handling in a large scale system in which the entire state of the task in execution almost certainly had to be preserved and for which considerable analysis may be needed to determine what to do following an interrupt.

The effect of boundedness both in restricting the problems of implementation of a system and in simplifying communication between the people involved in the implementation should also be noted. For example, an information retrieval subsystem, handling the problems of indexing and retrieval of records, might be designed and implemented without any concern about the details of manipulating the I/O devices.

2.4.2 There are also disadvantages in the subsystem design approach, the most important of which is an intercommunication problem. Particularly when subsystems have overlapping scopes or duplicate capabilities the number of inter-connections, needed to obtain efficient operation, begins to increase. Mr. Clark considered a subsystem, the function of which was to manage information on a disc system, consisting of a controller and a number of drives. He pointed out that, in a system containing two such subsystems, an interesting problem arises when there is a requirement to build an index, interpreted by one subsystem, to information

residing in the other. A special connection between the subsystems must be provided to permit this or alternatively the subsystem must reply that wanted information is not present, in which case the other can be tried.

A second disadvantage is the specialization of 'componentry' which can result by designing subsystems. For example, a number of identical CPU's in a multi-processor pool might be a better solution to the design of a particular system, providing some of the advantages of restricted context without the disadvantages of several pieces of specialized hardware to control specific operations.

2.5 Hardware alternatives in I/O systems design

The speaker indicated that, in his view, this material should not be presented only to those interested in the engineering aspects of system design. Programmers, being closer to the users' view of systems, will be increasingly involved in the specification of the architecture of future systems.

2.5.1 The variety of channel designs available may be made apparent by a hierarchical presentation of:

1. the simplest channel which is simply a port to memory, permitting the transfer of one byte from a device;
2. channels which can assemble bytes into a word;
3. channels that can increment memory addresses and thus store messages;

and 4. at the highest level, the intelligent channel or I/O processor which can execute a whole sequence of commands.

Intelligent channels may be sub-divided into two classes, those having their own memory to act as a data buffer and those which simply exert control over main memory. Cost attributes and the methods of use are quite different for these two classes. A sufficient variety of channels have been constructed so that there should be adequate criteria for choice of type; however this point has not been developed.

The speaker suggested that, in presenting hardware alternatives, one should emphasize the embarrassing situations which can arise if only part of the technology is considered. It is easy to build almost any logical structure into a machine, particularly when one considers micro-programmable machines which permit execution of long instruction sequences very cheaply, so that relatively inexpensive

machines can support quite complex channel capability. Large, fast machines present a very different picture as, relative to the cost of the machine, one would like to reduce channel costs since more of them are needed. This is impractical partly because the width of the memory bus is increased and partly because architecturally defined long control sequences have to be implemented in pure hardware.

2.5.2 Mr. Clark pointed out that the value of switching systems should be demonstrated; in particular it should be stressed that, by using large scale integrated circuits, switching will be inexpensive, the cost being determined mainly by programming support and the cost of the electrical connections that the switch implies.

2.5.3 It was suggested that, in a University course, a discussion of classes of devices, particularly drums and discs in the direct access storage device class, rather than individual devices is appropriate, particularly as there are now sufficient designs in the field to illustrate the different properties provided by different methods of control. An interesting, purely technological, distinction is in the recording format - whether record lengths are physically determined or recorded on the disc surface with the record. This difference has some important consequences. For example, scheduling algorithms appropriate to one may not be appropriate for the other; similarly, problems of channel utilization may occur in one situation but not in the other. We arrive at the latter method as the cheaper design by considering information density on the surface and the lower overheads between blocks when lengths are explicitly recorded. It does, however, require an intelligent controller looking at the surface of the disc for a period longer than is needed to read the record.

2.5.4 Problems inherent in the nature of electrical signals in relation to the initiation or termination of I/O operations and in relation to the achievement of high data rates are important but not usually very well understood. For example, the START I/O instruction in IBM S/360 machines involves communication with a device which returns a status code before completion of the instruction. On a very fast machine, communication with a device 100 ft. away would block the execution of some thousands of instructions. The START I/O instruction would consequently increase the cost of I/O scheduling out of all proportion; nevertheless, to maintain high I/O throughput, it is necessary to arrange close communication between devices and the CPU in order that the CPU can reflect its requirements.

The point was made that, with increasing record densities, record lengths will soon become so short that the time to read information from a disc may become less than the propagation time along the wire from the device. As a result, physical displacement of devices from the CPU may become more important in determining system throughput than the displacement of components in the CPU itself. The speaker noted that the present state of the art was close to this situation now.

2.6 I/O Scheduling

Although I/O scheduling is usually a small part of an operating system, Mr. Clark pointed out that it was quite complex when the problems of error recovery, pathfinding and the interactions arising in priority scheduling schemes using both I/O and CPU priorities were considered. He noted that I/O schedulers became quite different in the presence of hardware pathfinding, and suggested that it would be interesting to try and characterize the cost/performance 'trade-offs' of hardware versus software pathfinding in order to assist in making an appropriate choice.

3. System Balance

In this section the speaker discussed the problems of system balance in greater detail, suggesting that it would serve as an example of how the previous topics could be expanded for more detailed treatment. He suggested that system balance was a subject of much current interest with CPU performance outstripping that of I/O. The fact that I/O equipment costs outweigh those of the CPU in a system emphasises the importance of the understanding of data and I/O management. He pointed out that it could be shown that it is really I/O that prescribes large main memories and that the cost must be included with I/O costs.

3.1 Environment

In looking at memory requirements relative to CPU rates, several assumptions were made by the speaker, some implicit but fairly obvious. Statistics obtained from past programs on several machines were used; however, Mr. Clark indicated that these might not be valid in future. Indeed he would prefer some of them not to be! The system chosen by the speaker to be analyzed was:

1. a multi-programming system with a high order of multi-programming, using fast CPU's to provide an instruction rate of 10^9 instructions per second, which, by projection, would be practical within 15 years;

2. a partly interactive system, implying that I/O demands would increase faster than CPU power increases: (for example, extracting a 'record' from an inverted file may require 10-20 accesses. It was suggested that the computer's capability as a monitoring device would be more fully utilized by then, which would also increase access rates);

and 3. a system using direct access devices the speed of which may change in the future, although their general characteristics are likely to remain the same.

3.2 CPU vs. I/O speed

From the above assumptions it is clear that the CPU speed will far outstrip I/O performance; thus, a program loop in which, in say 1960, all I/O was fully overlapped by CPU processing will on such a machine provide negligible CPU activity per I/O request.

3.3 The System Model

The model to be considered views main memory as a buffer to handle access delays, rather than as a logical unit. Since I/O is so slow there will be a queue, conceptually between memory and the direct access storage.

One can assume:

1. a large enough order of multi-programming;

2. an infinite memory;

and 3. an infinite number of I/O devices with realistic access times and costs.

These assumptions give a rate of insertion into the queue limited by the CPU and a holding time appropriate to the I/O device. The objective is to determine the proportion of memory needed by the system to hold inactive programs and their data which will completely buffer the CPU utilisation during an I/O access in the originally active program. Alternatively, one might fix the memory size, which would determine the queue size (assuming perhaps that all programs are of the same size), and determine the optimum CPU rate.

Development of analyses along these lines would be a very valuable part of a course and with suitable assumptions and parameterizations should not be too difficult to achieve.

3.4 The Queue Elements

Each queue entry represents an area of store awaiting an I/O request. The context of the I/O request comprises:

1. the program awaiting the request which will occupy not less than about 2K bytes (possibly allowance should be made for having this shared by several users);

2. workspace for the I/O request which can vary from zero for non re-enterable codes to more than 2K bytes;

and 3. some information showing the relation of the program to the system which will occupy negligible space.

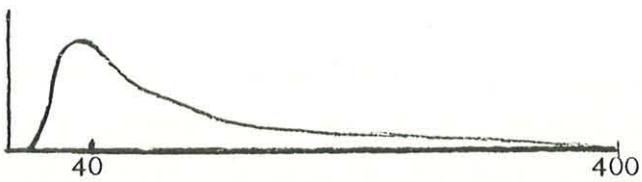
A lower bound of 4K bytes for the context in the queue is valid -- the American Airlines SABRE system approximates this. For an upper bound, a fairly arbitrary choice may be made. The speaker suggested that 256K bytes would be reasonable for Fortran H, and, in time-sharing systems, that figures in the range 64-128K bytes would be acceptable.

3.5 Relation between CPU rates and I/O demand

The ratio:

$$\frac{\text{executed instruction bits}}{\text{accessed I/O bits}}$$

has been measured, by program tracing, over many programs from different application areas and on machines of widely different architecture. These measurements lie on a distribution peaking at about 40 and becoming insignificant at about 20 and 400; thus the figure 40 forms a good design point, implying that a machine with a 20 bit instruction requires about half a bit of I/O per instruction executed. Mr. Clark stressed that these figures were relevant only to direct access I/O with programs efficiently utilising the resources of the system, and noted in passing that, in economic terms, an in-core compiler was usually not the most effective in a system with an extensive storage hierarchy.



He stressed that the figure of about 40 seemed very widespread and suggested that if it was possible to build larger systems, without loss of efficiency, then the variations from this figure would be reduced as the population of jobs increased; however, he felt that this last point was essentially a topic for research.

3.6 Relation between CPU rate and access rate

There is much less relevant information available for consideration of the ratio:

$$\frac{\text{executed instruction bits}}{\text{number of accesses}}$$

particularly as some of the programs analysed used data structures which are no longer appropriate: (e.g. a preponderence of 80 character records is no longer prevalent). It is not known whether results valid on IBM 360 obtain on other machines, and therefore it is perhaps only safe to say that the ratio is greater than 20K bytes/access! (i.e. on IBM 360 one is not likely to get below about 600 instructions per access, which is not surprising since I/O supervisor instructions are involved). A good design point would be in the region of 40K bytes, even though there are wide variations, and a system designed to work at that level should be a good interactive system.

3.7 Computing the memory needed for buffering

Given values for these two ratios, the previous data on program size, instruction execution rates and also some value for the queue holding time it is possible to compute the fraction of an infinite memory required as a buffer to keep the CPU busy. From the access frequency the order of multi-programming can be deduced which, with program sizes, allows computation of the memory requirements. This type of computation is not really valid since everything is made a simple function of everything else, but treating the various quantities as parameters and choosing them independently provides some insight.

Using current I/O equipment characteristics, a 2314 disc for example, this type of analysis suggests that about 2M bytes of buffer memory are needed per million instructions per second, a result which should be fairly easily reduced, possibly by paging. The problem of 'thrashing' in a paged memory system may be considered from the present viewpoint; then the context space assigned to each program in such a system is less than its natural value and the additional I/O which is induced in turn requires more buffering in order to utilise the CPU. Under such conditions there must be an optimum memory size relative to cost, which hopefully would be much less than the 2M bytes figure quoted. It is not immediately obvious that paging is beneficial, when one includes the overheads of paging and page tables in memory.

3.8 Determination of access times

Mr. Clark dealt with the parameters determining access time as follows:

3.8.1 Device characteristics.

The seek time can be adjusted considerably by different strategies but it was contended that this is not as important a factor as is the period of rotation. Another factor is the fraction of the total number of tracks covered by heads - the 'coverage fraction' - and it should be noted that data transfer rates have relatively little effect, compared to that of rotational latency, since most records are short.

3.8.2 The nature of the request.

In an I/O context this would be a read or a write and their times are equivalent, but in an information system one should consider both the record length and the indexing involved which may generate additional accesses or even cause volume changes.

3.8.3 The despatching strategy.

Various alternatives are available, Mr. Clark mentioned specifically:

1. FIFO, which leads to an unstable queue; since two requests take about twice as long as one, it is clearly unsatisfactory;
2. ordering seeks, where the object is to despatch so that arm motion is minimised;

and 3. in positional queueing, where the known angular position of a record relative to a reference point on the direct access device surface is used to reduce rotational delays. This strategy is interesting since for the first time it permits access times commensurate with data rates; its main application so far has been in paging drum designs and it is not yet in widespread use with discs.

3.8.4 Address distribution to the external device.

It was suggested that the parameters of interest would be the distribution of addresses over channels, controllers, devices, cylinders and tracks. In this context the known exponential distribution of popularity of addresses (after re-labelling) is relevant.

Mr. Clark pointed out the feasibility of constructing a simulation model here. After tracing sufficient programs one would choose a multi-programming strategy to merge addresses and look at the way the system behaves. The only difficulty in the simulation might lie in accommodating the access paths available.

3.8.5 Access paths available.

One might have an access matrix which attaches any device to any controller to any channel and while this can still be simulated, it makes the results more difficult to analyse.

This simple approach to system balance is valid and can be rendered in exact form, the parameters needed to obtain results being either known or determinable.

3.9 Factors related to system design

Finally the speaker discussed a number of system design features which affect system balance.

3.9.1 The cost of an access is determined by:

1. The cost of the I/O equipment. Here the cost of storage associated with the I/O operation is excluded and one considers only the cost of the data flow components, controllers and the time they are used, switches, ports, etc.

and 2. The context space-holding time product. The component of memory utilised by the CPU in any interval when it is connected to one program is very small. The rest is really I/O buffering and should be considered as such.

3.9.2 The context space-holding time product can be reduced by writing programs so that as many accesses are exposed as early as possible permitting techniques like positional queueing to be effective; historically this represents a complete reversal of former practice when, due to lack of parallelism in I/O devices, the motivation was to distribute accesses through a program.

3.9.3 Contention and other factors which tend to decrease efficiency with increasing order of multi-programming.

Contention need not be a problem if it is taken into account in the system design. For example, a good queueing and despatching system together with an elaborate switching system may yield very little if the address distribution is biased so that the same device, same controller and same channel are selected, which is precisely what happens in most current operating systems. Distributing data over devices deals with this adverse address distribution but it does not attack the problem of the biased popularity of addresses which requires the use of memory as a buffer holding data (as opposed to context) as is done with the

IBM 360/85. In this situation, the number of true accesses drops and so, consequently, does the number of context cells needed which in turn implies a decrease in memory size. Clearly there is an optimum.

Discussion

Following a remark by Dr. Browning concerning the alternatives of intelligent channel or CPU control, Mr. Clark pointed out that if the channel had private memory situated physically very close to the I/O device, there were access algorithms available which would take advantage of this known short distance. He cited the example of the positioning of records constituting successive levels of an index on a disc in such a way that, when they were accessed in sequence, there was minimal delay. He pointed out also that the insertion of the 'memory' of such a controller in the data path raised problems when attempting to maintain a continuous flow of data from the I/O devices. Similarly, there was the problem of the issue, by the CPU, of a further READ statement to the controller whilst it was passing an already retrieved record from its private memory to the main memory. Mr. Clark pointed out that this was only one of the problems occurring when two asynchronous controls co-operated and stressed that any attempts to increase the capability of the channel would increase the need for communication with the CPU giving rise to the necessity for multi-programming in that channel.

Professor Dennis pointed out that it was difficult for the designer of the logical data management system, in an implicit I/O environment to work without a knowledge of I/O devices and Professor Randell suggested that, by using explicit I/O, the user supplied the system with much more implicit information, and indicated that the control would need to be achieved in a rather different way as I/O became more implicit. In reply the comment was made that, to balance the exponential rate of improvement in CPU performance with a linear improvement of device performance in time, it would be necessary for the system to take full control of data organizations on direct access devices and that physical sequential data organizations could no longer be tolerated.

Techniques which could be applied to offset the merely linear improvement in device performance were discussed and in particular a technique was described, based on the observation, in disc address tracing that, over intervals of about one minute, systems with a large enough population of jobs exhibited a frequency of access to a population of disc addresses which was distributed exponentially (after

suitable relabelling of addresses) thus giving rise to the possibility of the reduction of accesses made to the disc for the frequently used addresses, given a sufficiently large buffer area.

The use of the ratio of real to virtual memory in paging systems as a system performance parameter was raised as opposed to the relation between CPU speed and the real memory available to it. It appeared that the former was used more in estimating paging requirements (e.g. the possibility of exceeding paging drum capacity) than as a parameter in balancing a system.

Dr. McKenzie raised the problem of data management schemes in the small system.

The data management schemes discussed in the talk were just feasible on systems comparable with the IBM 360/65 and apply well only to larger systems. Below this level the following recommendations were made regarding future small systems:

1. Use of a very limited amount of buffering to reduce accesses;
2. The need for simplicity must be emphasised;
3. The use of late binding, rather than complete pre-specification, since it is needed for interactive systems and typically costs nothing;
4. The use of table driven and interpretive techniques for binding (e.g. at OPEN time). The conventional macro system provided today in large systems is too expensive on small machines; e.g. the fetching of library routines at OPEN time in OS is too costly on small machines. In consequence, everything is opened only once and data base management must be undertaken as the price.

The elimination of pre-specification should extend not only to details of what has to be done but also to the data the structures used to contain this information. Although this information must be kept in the system, it is not always necessary to involve the user in it.

Professor Randell requested more details of the material to be presented on I/O interrupt handling - in particular on alternatives. The general comment was made that particularly during a high flux of I/O interrupts, there are a number which do not need immediate CPU attention, others are not particularly appropriate to CPU processing because of requirements that the CPU does not provide. A queueing system and some treatment of them in the I/O area itself

would be an attractive alternative.

The techniques for processing interrupts can be presented hierarchically as follows.

1. A trap system. Almost no context of the CPU is saved. A simple branch to a fixed location occurs.
2. Consider a PSW scheme where some but not all of the context is stored. In this category one can include schemes which branch to a different location depending on interrupt type.
3. Finally, a scheme in which the entire status of a task is queued automatically and another started automatically.

There is also a need for the ability to enqueue items without the overhead of task switching. The locking of the queue during the change implies that it must be done by the system, but a simpler mechanism than those currently in use should be available for this.