SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 1, 269-277 (1971)

The Kent On-Line System

P. J. BROWN
Computing Laboratory, University of Kent at Canterbury, England

SUMMARY

KOS is a simple multi-access operating system that is particularly geared to the needs of teaching and research. It is fully conversational in that it allows interaction between user and program at all times, and it supports a variety of teaching packages, compilers, application programs, etc.

KEY WORDS On-line Operating system Pure procedure

INTRODUCTION

The Kent on-line system, which runs on the ICL 4130, is an open-ended multi-language operating system which provides fully conversational console working at a minimum cost. This is achieved by making maximum use of pure procedures. The system is being developed at the University of Kent at Canterbury on behalf of all the universities with an ICL 4100 as their main service machine.

When design work started on the Kent on-line system (KOS), there were, as is usual in such exercises, many design requirements that had to be considered. However, one requirement took precedence over all the others. This was that the system be fully conversational. In other words, throughout the operation of the system and, most important, of all the compilers and packages that were to run under it, it should be possible for the console user to type instructions and ask questions of the system and vice versa. (See Reference 1 for a fuller discussion of this.) The reason for making this requirement paramount was that it was felt that the consoles must extend the capabilities of the machine by providing facilities that could not possibly be provided by batch processing; the consoles were not simply to provide a more convenient way of running programs in the batch (as, for example, the system described by Atkinson and co-workers[2])—though this approach has considerable merit and can be very popular with users—nor to provide a means for a privileged few to queue-jump.

There is a very wide potential market in any educational or research organization for fully conversational console working, in particular for teaching, simulation and application packages. Participation is an excellent way of learning. If a student was, for example, trying to understand matrix arithmetic, nothing could be better for him than to go and sit at a console and experiment with a matrix package. The student could type in his matrices, transpose, add, invert them and thereby get a feel for the way things worked. He would, of course, make lots of mistakes, which the matrix package would diagnose, and it would be by these mistakes, more than anything else, that he would learn. Ideally, the matrix package should be designed such that, as well as catering for the novice, it could be shared by users with production jobs to do. It is easy to think of a host of similar applications in other

Received 30 April 1971

disciplines; for example, the simulated salmon fishery described by Zinn and Hesselbart[3] is instructive to potential managers and, more generally, simulated models are invaluable for any kind of design work. Furthermore, there is no better way of learning to program than by using an interactive compiler.

This, therefore, is the kind of market at which KOS was aimed. The problem was to provide fully conversational working cheaply, since KOS was to be run on university service machines where the batch processing work could not be allowed to suffer. In solving this problem, certain features of the hardware of the machine to be used, the ICL 4130, played a major part.

THE HARDWARE

Broadly speaking, the ICL 4130 is a typical medium-sized scientific computer of its generation. It has a 24-bit word and the store cycle time can be either six microseconds or two microseconds. All the universities for which KOS is being developed have roughly similar configurations. Core store sizes are 32K or 64K words; all machines have, or are scheduled to have, a fairly comprehensive set of I/O devices; this includes exchangeable disk units. Each disk pack has a capacity of one million words. Most installations are planning to have on-line consoles and some already have them; in most cases the consoles are connected directly to the 4130 rather than to a satellite computer.

The 4130 has two modes of operation, namely 'executive mode' and 'slave mode'. The latter, however, has been completely ignored until very recently and virtually all programs, whether written by the manufacturers or by users, have run in executive mode. The slave mode of operation uses the well-known base and range register technique. There is an important sub-mode of slave mode, called *pure procedure mode.* (The manufacturers call it *common program mode.*) This has influenced the design of KOS more than any other hardware feature. In pure procedure mode the program instruction counter is absolute and it is only data references that are subject to the addition of the base. Using this hardware, a single program can control any number of slaves. A slave is simply an area of workspace, and the slave that is active at any one time is determined by the setting of the base and range registers.

There is a further hardware feature that allows for 'pure constants'. Each pure procedure can have its own data area which contains all true constants, i.e. all data whose value is set at load time and not subsequently changed. When using pure procedures, it is best to place tables and error messages in this data area, so that there will only be a single copy of them rather than one copy in each slave area.

The pure procedure mode of the hardware and its ease of use radically changes the economics of time sharing for the 4130. For multi-access work the 4130 is at its most efficient when there are one or more pure procedures, remaining permanently in store while they are in use, being shared by several jobs. For example, there might be two pure procedures in use, a compiler and an application package; three console users and a card-to-printer job might be using the compiler, and four other console users might be using the application package. All should enjoy excellent response time. Moreover, if the pure procedures are always resident in core there is no overhead in providing fully conversational working.

The design requirements of KOS were therefore met by making it an operating system that provided a convenient environment for writing, testing and running pure procedures. A pure procedure, or a suite of related pure procedures, that runs under KOS is called a *sub-system.*

CURRENT STATUS

Development work on KOS commenced in late 1968 and a preliminary version was made available to the public in May 1969. By February 1970 all the features described in this paper had been implemented and a regular user service set up. In December 1969 the first fully documented release of KOS to outside users was made.

The emphasis of development has now shifted from the system itself to the provision of sub-systems. A number of sub-systems are already working and went out with the first release. These included a compiler/interpreter for Dartmouth BASIC[4], a macro processor,[5] a simulated desk calculator[6] and a multiple regression teaching/application package.[7] The last-named has been used in teaching a statistics course.

As time goes by it is hoped that KOS will continually be augmented by new sub-systems developed not only at Kent but at other installations as well. Indeed, KOS will have failed if this does not happen. Particularly pleasing, therefore, is the development of the BASIC compiler, which is a joint venture between Kent and the University College of North Wales, and the development of an assembler/debugging system by W. D. Shepherd of Portsmouth Polytechnic.

RESOURCES

The scale of available man-power resources has been a major consideration in the design of KOS. The Computing Laboratory of the University of Kent at Canterbury is currently small, and is only able, without prejudicing its other commitments, to allocate the equivalent of about three full-time people to the KOS system and its sub-systems. In addition, an extra research associate and secretarial assistance have been financed by the Computer Board.

Nevertheless, since the implementation of efficient general-purpose multi-access systems has defeated organizations with many times the resources, in both quality and quantity, of the KOS project, it has been a policy to avoid over-ambitiousness. In particular, the natural urge to re-write everyone else's software has been resisted, and, where applicable, the manufacturer's software has been used.

KOS AS THE USER SEES IT

It is not worth giving a complete description of the features of KOS since a number of them follow familiar principles pioneered by earlier operating systems. Instead, details will be given only of those features which might merit special interest; other features will be omitted or glossed over briefly.

In this latter category come editing and disk filing. KOS, in fact, uses the manufacturer's disk filing system in order to maintain compatibility and interchangeability with the batch. There is also provision for small in-core files. These will be described later. During editing, lines can be specified by a line count (e.g. line 209), by contents (e.g. the line starting with '[XXX]'), by relative line count (e.g. 11 lines further on) or by a combination of all these (e.g. 11 lines beyond the first line starting with '[XXX]' after line 209). Editing is fully conversational and thus the user can, prior to changing a line, examine it to make sure he has got the right one.

USE OF I/O DEVICES

Each of the job streams that KOS controls has a default input device and a default output device assigned to it. Normally during a KOS session most of the job streams will be attached to consoles, the same console being used as the default input device and the default output device. In this case KOS is said to be in *conversational mode*. However, it is quite possible to have non-conversational job streams such as card-to-printer or paper tape reader to paper tape punch. Apart from small differences, mainly concerned with physical properties of devices, it makes no difference to the working of KOS whether it is in conversational mode or not. Thus production runs using sub-systems such as the BASIC compiler can be performed in a batch mode. In such non-conversational cases a job will usually involve only a short sequence of commands; in conversational mode a job will normally encompass the entire stay of a user at a console.

The default devices for a job stream are said to be *owned*. Any device that is not owned is free to be *borrowed* by any job that wants it. The purpose of the owned devices is that if a job is suddenly stopped, for example, because of a filing error or because the user has pressed the break key, there are always devices available on which KOS can output the appropriate message and obtain the next command.

Each job has two types of input: commands and data. Commands commence with an ampersand. For consoles, KOS types an ampersand when it is ready for the user to type a command; for other devices the user supplies the ampersand. Commands come from the *command device*. This is usually the same as the default input device but it can be altered by commands such as

```
&COMMANDS FROM CARDS
```

or

```
&COMMANDS FROM JFILE. PJB.
```

(In KOS terminology the word 'device' means either a physical I/O device or a file, i.e. what some operating systems call a 'stream'.)

Some commands have associated data. For example, the data to the BASIC command is a program to be compiled and the data to some of the editing commands is the text to be inserted. Data comes from the *data device*. The data device only logically exists during the execution of a command that requires data. By default the data device is the same as the command device, but this may be changed by adding a suffix to a command, for example

```
&BASIC FROM CONSOLE 4
```

For console input KOS types a colon when it is ready to receive a line of data. The use of a colon rather than an ampersand is to prevent a confused user typing data when he thinks he is typing a command. This helps to prevent some of the ludicrous situations that often arise when users get at variance with the system. If a console user wishes to return from data to command status he must type the data terminator (a single full stop) or, more drastically, press the break key.

For input from other devices, data has no special prefix. It is terminated by the next command, i.e. the next line starting with an ampersand. This exactly mirrors the 4130 batch operating systems, to which many KOS users will be accustomed; here control cards, which are the equivalent of commands in KOS, are identified by an initial ampersand.

As for input, so for output. There are two output devices, namely the *message device* and the *results device*. The message device always exists and is usually the same as the default output device. The results device is created for those commands that produce answers. Results can be sent to a device other than the message device by adding a suffix to a command, for example

```
&RUN TO JFILE 3
&CHESS FROM CONSOLE 1 TO CONSOLE 2
```

(RUN is the command to run a BASIC program. The CHESS command is hypothetical.)

With these facilities a console user can do work where all or most of his data is on disk files and all or most of his results are to go to disk files, but he can, if he wishes, have all error and informatory messages sent to his console.

In practice this I/O scheme has so far worked very well. One can, however, imagine possible future sub-systems which would be seriously hindered by the restriction that there can be, at any one time, only two different input devices and two different output devices.

QUESTION-AND-ANSWERS

A further feature of KOS I/O is the *question-and-answer*. The basic purpose of this is to allow a sub-system to ask a console user a question and get a reply on the same line, for example

NUMBER OF POINTS = 36

OPTIONS REQUIRED =

where the text up to and including the equals sign is the question and is typed by KOS and the remainder, which in the second case is null, is the answer and is supplied by the user.

However, question-and-answers have been designed so that they work just as well in non-coversational mode, and indeed, serve an additional function in this situation. (The question-and-answer is, in fact, currently the only feature of KOS which works in a significantly different manner depending on the mode of operation.) Assume, for example, that card input is being used. In this case the user would supply, at the point in the deck where KOS would ask the question, a card containing the question and his answer. For example

NUMBER OF POINTS = 29

When KOS executes a question-and-answer from cards it reads the next card and tries to match the text at the start of the card with its intended question. If it succeeds, all is well. If the match fails, however, it is not necessarily an error. KOS then 'backspaces' the card reader and tells its sub-system that the required question-and-answer is not there. The sub-system can then take one of three courses:

(1) It can try to match the card with another question.
(2) It can treat the situation as an error.
(3) It can assume some default option and proceed normally.

Course (3) would probably be taken for a question-and-answer such as the 'OPTIONS REQUIRED = ' example above. In this case the question-and-answer serves the same purpose as optional keyword parameters do in IBM macros and operating systems.[8]

JOB FILES

As well as disk files, KOS allows for files to be kept in core. These are called *job files* since they are deleted at the end of each job. In most cases job files must be small as users will normally have very limited core workspace; they are useful for temporary storage of small amounts of data, such as a student might need for a tutorial package. Editing of job files is destructive, i.e. only one copy of each file is maintained and as the file is edited it continually changes. (The user can, if necessary, explicitly make a copy of a file before editing it.) Job files are therefore very useful for short files that are likely to be extensively edited during a console session, since they provide fast access and fast editing and relieve the necessity of creating a new version of the file after each edit and remembering which version is which. Job files can be copied on to disk at any time when it is felt that a more permanent record is needed.

HIERARCHY OF COMMANDS

A KOS sub-system is entered by typing its name as a command. In this case KOS exits from the current sub-system, if any, and enters the named one. Initially no sub-system is in control. KOS commands consist of two types: *global commands* and *supplementary commands*. Global commands are part of the KOS system itself, and may be used anywhere, whether a sub-system is in control or not. Typical global commands are commands for entering sub-systems, for changing devices, for editing job files and for enquiring about the environment.

When a sub-system is entered it can return to command status within itself and can augment the global KOS commands with its own supplementary commands. For example, the BASIC sub-system has supplementary commands such as PROG (amend program), SCR (scratch everything), CONT (continue after break), LIST and RUN. These commands do not exist outside BASIC. It is also possible, while remaining in command status within BASIC, to execute any of the global KOS commands.

This hierarchical scheme has the advantage of insulating the non-user of a sub-system from its supplementary commands while still allowing the full repertoire of global KOS commands to be used within any sub-system.

THE USE OF DEFAULT OPTIONS

To minimize the typing necessary for simple-minded users, KOS makes considerable use of default options. For example, if nothing is said to the contrary, the default I/O devices are used throughout. Similarly, there are concepts of the current line and the current file, which are used by default. Similar concepts to these are used in the Cambridge Multiple-Access System.[9] In this way KOS has followed PL/I,[10] while hopefully avoiding some of the controversial situations where it is not sufficiently manifest what the default option should be.

USE OF STORE

The number and nature of the KOS job streams to be run and the amount of workspace allocated to each is fixed when a KOS session starts and cannot be changed during the session. Pure procedures are loaded dynamically when they are needed and are automatically cancelled when they are not in use. Core storage is divided into fixed partitions. The whole philosophy of pure procedures is against them being swapped in and out of core while in use, but KOS currently has the additional restriction that slave workspace areas also remain in core throughout. Typically, these workspace areas are about 2K for console job streams and perhaps 4–6K for a card-to-printer job stream.

A map of the current store layout is given in Figure 1.

INTERFACE WITH MANUFACTURER'S OPERATING SYSTEM

Currently, virtually all the intending users of KOS use the manufacturer's standard single job stream operating systems for all their production work. We will use the term ICLBATCH for these operating systems. (The manufacturers have recently introduced a

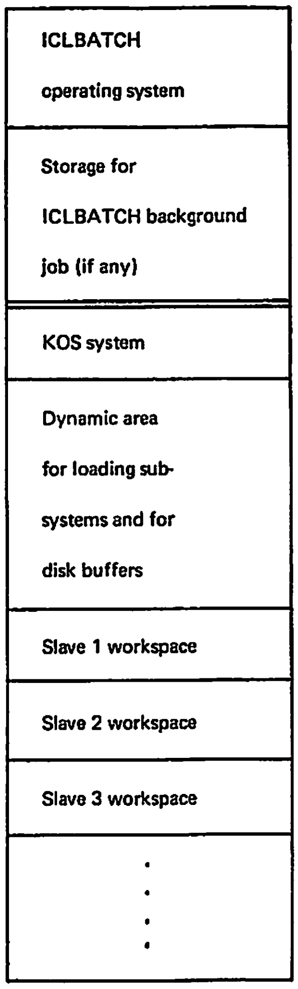

Figure 1. Layout of core storage when KOS is running. The relative sizes of the boxes above bear no particular relationship to the relative sizes of the storage areas they represent

new multi-programming operating system. Currently this is little used, but it will, no doubt, gradually find wider usage. KOS will eventually run under this system and certain of its features will need to change as a result. In this paper, however, attention will be restricted to the current implementation of KOS.)

The following features of KOS have been introduced to minimize the inconvenience to ICLBATCH work while KOS is running:

(1) An ICLBATCH batch background job stream can be run against KOS. In this case ICLBATCH, rather than any KOS job stream, owns the card reader and the printer.

(2) KOS runs under ICLBATCH. A KOS session is initiated from ICLBATCH using ordinary control cards and at the end of a KOS session control is handed back to ICLBATCH. As far as ICLBATCH is concerned, a job that creates a KOS session looks no different from any other job. There is a completely smooth transition from

ICLBATCH to KOS and vice versa and no time is wasted by the need for the operator to intervene. When KOS is in control, of course, it places so many tentacles round ICLBATCH that KOS is effectively the boss. KOS can even re-enter ICLBATCH and leave it to run a background job as mentioned in (1).

(3) As mentioned earlier, both ICLBATCH and KOS use the same disk filing system.

TESTING AND RUNNING

A consequence of (2) above is that test runs of new KOS sub-systems can be run as ordinary batch runs. This is done as follows: A test deck for the sub-system is prepared on cards. (This may also make use of disk files.) Since question-and-answers can work from cards, comprehensive test decks can be built up even for sub-systems which are so interactive that they would not normally be used with card input. The test deck is preceded by ICLBATCH control cards to set up a KOS session with a single card-to-printer job stream. A command to terminate the KOS session is placed at the end of the test deck. The entire deck is then run as an ordinary ICLBATCH job; if the deck is short it may only take a few seconds of computer time to run it. If the program under test goes awry it will almost inevitably produce a trap, probably for a store reference out of range, and at this point an automatic post-mortem program comes into effect.

INTERFACE BETWEEN KOS AND ITS SUB-SYSTEMS

KOS consists of some modules that run in executive mode and some that run in slave mode; the former include modules for job stream creation and scheduling and for allocation of I/O devices; the slave mode modules, which are themselves pure procedures, provide a set of routines, called UTILIT routines, for use by sub-systems. These UTILIT routines call executive mode routines when they need them.

It is through the UTILIT routines and only through these, that sub-systems communicate with KOS. UTILIT routines are called by ordinary sub-routine calls. All I/O is done through UTILIT routines. To a sub-system, all I/O appears to be device independent. A sub-system never knows or needs to know the number and nature of the jobs that are using it. Hence a sub-system is written in exactly the same way as any more conventional program is written.

In addition to I/O, the UTILIT routines also provide a large number of other functions that are, or may be, needed by sub-systems. Of these, the most interesting is probably the routine UDECODE. This is an extremely elementary table-driven syntax analyser (though not, it seems, quite so elementary that it is easy to persuade people to use it). If, for example, a sub-system wishes to add some supplementary commands to KOS, it supplies UDECODE with a table for recognizing and analysing these commands and their arguments.

Although the size of each slave workspace area is fixed once and for all at the beginning of each KOS session, storage within this area is allocated dynamically. It is shared between buffers, job files and sub-system workspace. A fixed two hundred or so words of slave workspace are reserved as directly addressible storage for the current sub-system. If a sub-system needs more store than this it must ask for it dynamically using UTILIT routines.

It is not, in general, very difficult to convert an existing program in assembly language written for, say, the ICLBATCH operating system, to make it run as a pure procedure under KOS. The main difficulty is converting the operating system interface; for a highly operating system dependent piece of code, such as a compiler, this is very hard to do.

FURTHER DOCUMENTATION

Full descriptions of KOS as it appears to the user, the computer manager and the sub-system writer are given respectively by Brown and Brown,[11] Brown[12] and Brown.[13] User manuals for sub-systems are also available as given in the references.

ACKNOWLEDGEMENTS

A good proportion of the KOS system was coded by Mrs. H. Brown. The routine to control the consoles, which was, of course, vital to the utility of KOS, was provided by Spratt and co-workers.[14] In addition, since KOS stands or falls by the sub-systems it provides, the sub-system writers already cited have played a major part.

Finally, the Computing Laboratory of the University of Kent at Canterbury is very grateful to the Computer Board and to ICL for their assistance and co-operation.

REFERENCES

1. M. V. Wilkes, *Time-sharing Computer Systems*, Macdonald, London, 1968.
2. M. P. Atkinson, A. M. Lister and A. J. T. Collin, 'Multi-access facilities in a single stream batch processing system', *Comput. Bull.* **14**, 75–77 (1970).
3. K. L. Zinn and J. C. Hesselbart, 'Interactive programming languages adapted for instructional use of computers', *Int. Symp. Man-Machine Systems*, Cambridge, 1969.
4. S. E. Binns, W. H. Purvis and E. B. Spratt, *Kent On-line System: Compiler for a Subset of Dartmouth Basic*, Document KUSE/BASIC, Computing Laboratory, University of Kent at Canterbury, 1969.
5. P. J. Brown, *ML/I User's Manual, Appendix G: KOS Version of ML/I for the ICL 4130*, Computing Laboratory, University of Kent at Canterbury, 1969.
6. D. Glanville, *Kent On-line System: Desk Calculator*, Document KUSE/DESK, Computing Laboratory, University of Kent at Canterbury, 1969.
7. M. J. Garside, *Kent On-line System: MJGREG, General Stepwise Multiple Regression Program*, Document KUSE/MJGREG, Computing Laboratory, University of Kent at Canterbury, 1969.
8. D. N. Freeman, 'Macro language design for System/360', *IBM Systems Jl*, **5**, 63–77 (1966).
9. D. F. Hartley (Ed.), *The Cambridge Multiple Access System: User's Reference Manual*, University Mathematical Laboratory, Cambridge, 1968.
10. G. Radin and H. P. Rogoway, 'NPL: highlights of a new programming language', *Communs Ass. comput. Mach.* **8**, 9–17 (1965).
11. P. J. Brown and H. Brown, *Kent On-line System: User's Manual*, Document KUSE/AAA, Computing Laboratory, University of Kent at Canterbury, 1969.
12. P. J. Brown, *Kent On-line System: How to run KOS*, Document KOP/AAA, Computing Laboratory, University of Kent at Canterbury, 1969.
13. P. J. Brown, *Kent On-line System: Sub-system Writer's Manual*, Document KOS/AAA, Computing Laboratory, University of Kent at Canterbury, 1969.
14. E. B. Spratt, D. Glanville and J. Dobby, *Specification of a Device Routine for the Multi-Access Teleprinter Controller on the ICL 4100 System*, Document KOS/MATPCD, Computing Laboratory, University of Kent at Canterbury, 1969.