The Egdon system for the KDF9

By D. Burns, E. N. Hawkins, D. R. Judd and J. L. Venn*

An operating system based on the use of high-level languages and on the use of a discfile for program storage is described. The use of the discfile and the fact that only a small fraction of a program has to be recompiled when an error is corrected means that a very quick turnround is provided to users.

The Egdon† programming and operating system is designed for a medium to large computer installation in which a large proportion of time is occupied in program testing and in which most programs are written in high-level languages.

It is currently operating on two KDF9 computers each with the following configuration of equipment:

1 central processor with 32,768 48-bit words of core store.
8 English Electric–Leo–Marconi tape units.
1 Ampex TM4 (IBM-compatible) tape unit.
1 Data Products discfile (capacity 3·9 million words).
1 line printer.
1 card reader.
1 card punch.
1 paper tape reader.
1 paper tape punch.
1 monitor typewriter.

It can, however, be adapted to work with variations of this configuration.

The system is of a general type which though common in the U.S., is almost unknown to British manufacturers. Its main features may be summarized as follows:

1. Its unit of compilation is not the complete program but the routine.
2. It deals with a mixture of source languages.
3. It makes extensive use of a discfile for program storage.
4. Although it does not use conventional time-sharing it buffers input and output by the use of "pseudo-off-line" card reading, card punching, and printing.

The first of these features, the independent compilation of routines, is of great value in handling large programs. Each individual routine is compiled, not to its final "absolute binary" form but to "relocatable binary" (RLB), a partially compiled form in which the addresses of core locations have not yet been filled in and which can therefore be adapted to be obeyed from any position in the core store. Before a program is executed, the routines which compose it are assembled together and "relocated," i.e. their absolute addresses are filled in. The advantage of doing things this way is that if a small amendment is to be made to a program, only the routine which is to be amended has to be recompiled from the source language. The routines which have not changed are merely "relocated" as usual, which is much quicker than compilation.

Independent compilation is intimately associated with card input because if paper tape is the basic input medium, it is almost essential (because of the difficulty of amending paper tape) to hold a copy of the program in the system so that it can be updated regularly. Since it is undesirable to hold a program in both source language and RLB, the RLB is usually omitted and programs are compiled completely from source language each time they are executed. With cards no such difficulty exists. It is a simple matter to amend card packs, so the source language exists only on cards and the system merely holds the RLB form of a program. When an amendment is made a complete routine is input afresh from cards.

With regard to point 2, the mixing of source languages, these are at the moment EGTRAN (a dialect of FORTRAN II) and KDF9 User-code. ALGOL may be added later. Any given routine must be written in one and only one of the source languages, but EGTRAN routines may be freely interspersed with User-code routines. Both source languages are compiled to RLB, which serves as an interface language.

The use of the discfile for program storage as well as data is a central feature of the system. Not only are system programs and library subroutines held on the disc, which has been done before, but also some problem programs. We believe that in this respect Egdon is unique amongst conventional operating systems, although it is of course usual in multi-console systems.

The effect of using the disc, together with a fairly large core store and single-pass compilers, is to avoid completely the necessity for batch processing. This is usually necessary with non-disc systems, to avoid excessive searching time on magnetic tape. The Egdon system is a straight-through system, and jobs can be loaded into the card hopper one after the other, each

† Readers of Thomas Hardy will know that Egdon Heath corresponds roughly with Winfrith Heath, in Dorset, where the first KDF9 to use this sysem is installed.

* *English Electric–Leo–Marconi Computers Ltd., Kidsgrove, Stoke-on-Trent, Staffs.*

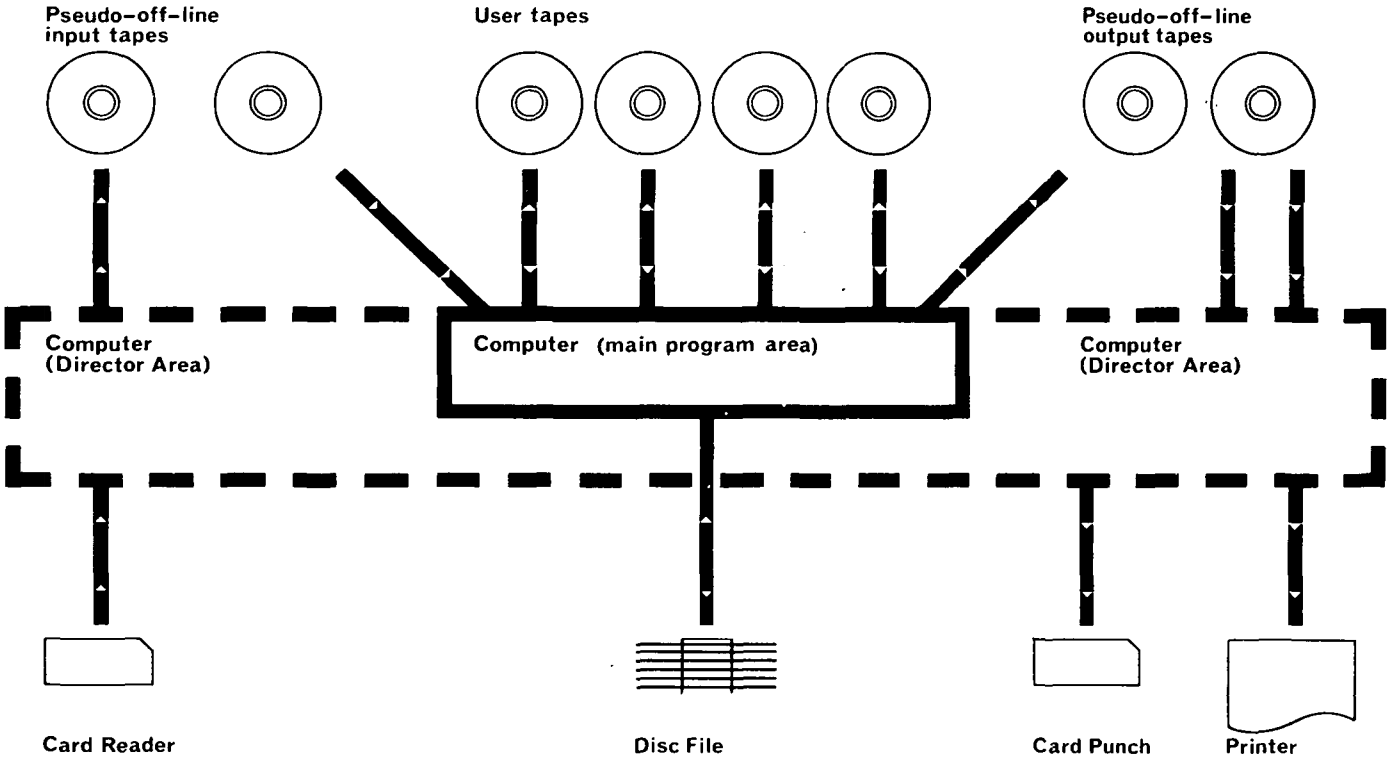

Fig. 1—The flow of information through the Egdon system

with its data with a minimum of operator intervention. The flow of information through the system is shown in **Fig. 1.**

The fourth distinctive feature, "pseudo-off-line" input and output, speeds input and output up considerably without affecting the "straight-through" nature of the system. Cards are read by the supervisory program (the Director) and the information they contain is copied on to magnetic tape quite independently of the job which is actually being run. The card images are read back from tape when the job they refer to is compiled or executed in its turn, so that all the information concerned with jobs that are currently being compiled or executed is read from magnetic tape, not from physical cards. Through-put speed is thereby increased, and if a card jam should occur time is not usually lost because the transcription process, not the running of jobs, is held up. The converse happens on output, information being written on tape at the time the job is executed and subsequently read back and printed or punched under Director control. These facilities are more fully described below under "Operating features".

The working of the system

The organization of the compilation and assembly of problem programs is done by a system program called Job Organizer, which is called into store by the supervisory program (Director) when work is to begin.

Let us consider first the case in which a card pack consists entirely of routines in source language (EGTRAN or User-code), together with control cards. The pack begins with control cards identifying the job and the segment and serving certain other special functions. (N.B. A *segment* is defined as the largest subdivision of a program held in core store at any one time during execution.) Job Organizer reads these control cards (or rather card images from magnetic tape) until it reaches the control card which immediately precedes the first routine. This card specifies which of the source languages the first routine is written in. Job Organizer then brings down from the disc the appropriate compiler for that language and transfers control to the compiler.

The compiler then reads the cards or images of the routine itself, and compiles the routine into RLB which it stores in the core store. Eventually a card is read which signifies the end of the routine. Control is then handed back to Job Organizer, which writes the RLB to an area of the disc called Job Assembly 1.

Job Organizer then deals with subsequent routines in the same way, except that if a routine is in the same language as the preceding routine, the compiler is not brought down from the disc because it is already in the core store.

When a complete segment has been compiled, Job Organizer automatically fetches from the disc all the library subroutines which are required, and copies them into the Job Assembly 1 area after the routines supplied by the user. If there are more segments, these are dealt with in the same way and stacked, each with its library subroutines, one after the other in the Job Assembly 1

area. All of these segments and their constituent routines are in RLB form, i.e. they have not yet been bound into a single program by having their absolute addresses filled in.

When the complete program has been written to the Job Assembly 1 area, Job Organizer calls a system program called *Relocator* which brings each segment in turn back into core store, "relocates" it, and writes it to another area of the disc called the Job Assembly 2 area. Each segment is now in absolute binary form ready to be obeyed. It is from the Job Assembly 2 area of the disc that segments are brought down during execution.

Assuming that execution is required, the first segment of the program is now brought down and entered. If the program uses card data, this will have been fed in on cards immediately after the program, and so will follow it immediately on the input tape. If magnetic tapes are required, details of these will have been given to the system by means of control cards in the job pack. If the right tapes have been loaded, the system will find them and allocate them to the program. If they have not been loaded, a message stating which tapes are required will be typed for the operator as soon as the program tries to use that tape. The program will be executed until failure occurs, until it terminates normally, or until it is terminated by the operator.

On termination, Job Organizer is brought down again and the whole cycle of events is repeated. If the previous job has failed to read all its data, this is read and ignored until a beginning-of-job card is found, so jobs may be loaded into the hopper one after another with no fear that one will interfere with another.

All cards except data cards are listed as processing takes place, via the pseudo-off-line output tape. Failure diagnostics are output in the same way, as of course are the user's results. There is also a facility for obtaining RLB versions of routines on punched cards, by preceding the routine by a special control card.

The above is the simplest pattern of the flow of jobs through the system. There are, however, variations on this. For example, routines may be presented to Job Organizer not in source language form but in the form of RLB cards. In this case Job Organizer does not, of course, have to bring down a compiler, but merely copies the RLB from the cards straight to the Job Assembly 1 area.

Alternatively, the program or part of it may be stored in the problem program area of the discfile. In this case a special control card is included in the job pack, and Job Organizer automatically brings down the program, which will already be in RLB form. A program segment may be composed of some routines from the disc and some from cards. If all the routine names are different, all the routines will be included. If a routine from cards has the same name as a routine from the disc, the one from cards has precedence. Thus it is possible to include modified versions of disc routines in the job pack. They will override the disc versions, without actually altering what is stored in the problem program area of the disc.

A further facility is that special control cards can be introduced in the job pack which cause a number of card images stored on the disc to be inserted into the input stream as though they had been read from the card reader. This is known as "data substitution". The card images which are inserted may be stored semi-permanently on the disc in the manner of library subroutines, or they may be put there temporarily during the assembly of a particular job by including them (with special control cards) near the beginning of the job pack itself. The latter method is only useful, of course, if the same sequence of cards has to be read several times in the course of a job.

The information which is stored semi-permanently on the disc (problem programs, library subroutines, library data substitution blocks) is put there in a special disc updating session which does not form part of normal running. The actual updating is done on magnetic tape. A copy of the contents of the disc is always held on magnetic tape for back-up purposes. In fact three separate tapes are used, part of the information being on each tape. The updating process is combined with copying—either tape-to-tape copying or disc-to-tape (not tape-to-disc). The updated version of the information then exists on magnetic tape. The disc itself is loaded from magnetic tape by a special loading program.

The three magnetic tapes used as a back-up can be used in an emergency as a disc simulator. The system is then considerably less efficient, but not absurdly so. The simulator also simulates the part of the disc which is available to the user, special overwriting techniques being used. Since there is no control over the way the user uses the disc, however, simulation of the user's area may be very inefficient.

Operating features

As has already been mentioned, both card input and card and printer output are normally done "pseudo-off-line," i.e. on input, cards are transcribed to magnetic tape by the Director quite independently of the problem program which is actually running, and the converse happens on output. Alternative modes of operation are available with direct input and output.

Basically two tape decks are needed for input and two for output. Taking input first, cards are read and a binary copy of the information they contain is written on to magnetic tape (each card being represented by a physical block on tape). "Binary" in this sense means that the exact pattern of holes is copied, each column being represented by 12 bits on tape. No processing at all is done at this stage. Apart from being inspected for certain patterns which indicate that the card is a directive to the pseudo-off-line process itself, the information on the cards is for the time being irrelevant. A tape that is being copied on to in this way is called a tape *u*.

While this is going on, a tape that has previously been produced in this way is being read back and is acting as an input to the central part of the system, i.e. the Job

Organizer, compilers or problem programs. A tape that is being read in this way is called a tape *v*.

A switch of functions between the two input tapes occurs when a tape *v* is exhausted. The tape *u* is then rewound and becomes a tape *v*, and tape *v* likewise is rewound and becomes a tape *u*. To avoid holding up calculation while tape *u* rewinds, a special control card can be inserted anywhere in the input pack which causes it to rewind and wait, so that reading can start as soon as the switch takes place.

At the beginning of a work session, there is of course no tape *v*. Reading takes place to tape *u* until 600 cards have been read, and a switch then takes place.

A further mode of operation is available in which no automatic switching takes place, to deal with the case where it is required to build up a stock of transcribed tapes for a run which reads large numbers of cards very quickly.

Pseudo-off-line output is a slightly more complicated feature. It deals not only with printer output but also with punched card output. Two tape decks are used, one on which information from the job which is currently running is being written (known as tape *x*), and one from which a tape which has previously been written in this way is being read back by Director and printed (known as tape *y*). Blocks are variable in size, each one normally representing the output of a single EGTRAN statement.

Items of information to be printed are mingled on magnetic tape with items to be punched on cards, the kind of information being indicated by a control word at the beginning of the tape block.

Items for output are also classified in other ways, according to whether they originate from the system itself (e.g. compiler diagnostics) or from the problem program, and, if it is problem program output, according to priority, i.e. whether it is "immediate" or "delayed" output (this facility is described in more detail later). When a tape *y* is "played back", the operator is able to indicate which classes of output are to be dealt with in this pass of the tape (except in "rush-hour" mode—see below). A special record written on the tape itself records which items have already been printed or punched, and which still remain to be done.

At the beginning of a pass of tape *y*, the operator might, for example, indicate that all "immediate" printer output should be produced, but nothing else. On a subsequent pass he might indicate that all previously unprinted printer output, and all card output, should be produced, and so on. A combination of options is available which enables the operator to print or punch all information which has not been printed or punched before.

Because a tape *y* may need more than one pass before all its information is printed and punched, it is not possible to switch tapes automatically as is done on input. The operating procedure varies slightly according to whether "rush-hour" or "production" mode is in operation, but the operator always has to unload the tape *y* and save it, and load a scratch tape for the new tape *x*.

In order to ensure quick turnround of results to users, a maximum running time and maximum amount of printing are fixed for each program. If the time limit is exceeded, the program is terminated (normal wind-up procedures of course being allowed). If the print limit is exceeded, all subsequent items sent to the pseudo-off-line tape are classified as "delayed", rather than "immediate" which is the normal classification.

Besides the normal "production" mode of operation in which time and print limits are fairly generous, there is also a mode called "rush-hour", designed to provide an extra rapid turnround on short jobs, in which time and print limits are small. In rush-hour mode, operators are allowed less freedom in the handling of pseudo-off-line print tapes than in production mode. For example, no printing other than immediate printing which has not been done before is allowed, and each *y* tape *must* be the previous *x* tape (i.e. an *x* tape cannot but put aside and printed later).

Apart from loading cards in the hopper and handling pseudo-off-line output, almost the only thing the operator has to do is the loading and unloading of magnetic tapes. He is aided in this by a tape control scheme incorporated in Director, which locates the correct reel no matter which deck it is loaded on, checks labels, takes automatic corrective action in the case of parity failure, and produces statistics of tape usage and performance. An interesting feature is that label blocks are accessible only to Director—they cannot be over-written by problem programs.

The discfile

The use of the discfile has certain interesting features. Like most discfiles, the KDF9 disc has the characteristic that the time taken to switch from one track to an adjacent track is, because of the time taken to move the positioner arm, considerably more than the time taken to switch to a totally different surface, assuming that no arm movement has to take place to bring the head into position on the new surface. The addressing of the disc is therefore arranged "cylindrically", i.e. so that sequential addresses, having passed along all the tracks accessible to one positioner, pass to the corresponding tracks on the next disc and thence to the corresponding tracks on other discs of the file, instead of passing to adjacent tracks on the same surface. Sequential addresses are thus arranged spatially along concentric "cylinders" rather than along plane surfaces. In fact, for a reason given below, the cylindrical addressing is not continued through all sixteen discs which compose the file, but stops short at the eighth disc. The file is thus divided into two halves, each half having cylindrical addressing within itself. No address in the first half of the numerical range of addresses will lie on the same disc as any address in the second half of the range. Since the positioner arms of the KDF9 can move independently (*not* all together like a comb) any area of consecutive addresses of less than a certain size (in fact 30,720 words) within each half of the addressing

scheme will be "movement-independent", i.e. once the arms have settled into position for accessing this area, any part of the area may be referred to without necessitating further arm-movement.

A further feature of the addressing scheme is that it optimizes as far as possible the use of fast and slow zones of the discfile. (Again like most discs, the KDF9 disc packs twice as much information in the outer tracks as in the inner ones, hence the transfer rate is twice as great.) To the programmer, an address consists of two parts, a "logical disc number" and a "sector number". Logical discs always begin at the beginning of a series of fast zone addresses, so that if a unit of information does not exceed a certain size (actually 2560 words) it can be stored entirely in a fast zone.

Disc storage is allocated in the following way at the time of writing, though this allocation may well be changed in the future. Areas are named in the order in which they occur.

AREA	APPROXIMATE SIZE (THOUSANDS OF WORDS)
Subroutine library	131
Library data substitution	131
Temporary data substitution	80
System programs	77
Job Assembly 2	230
Problem Programs	1317
User's area (also Job Assembly 1)	1966
	3932

It will be noticed that the user's area (the beginning of which is used between jobs for the Job Assembly 1 area) occupies the second half of the disc while everything else is in the first half. This means that no positioner arm which is used to access the Job Assembly 1 area is used for any other area used by the system. The reason for this is that during job organization, Job Assembly 1 is constantly being referred to. In the first place various items (e.g. library subroutines) are brought down from the first half of the disc and packed into Job Assembly 1. Later, at the relocation stage, chunks of programs are brought down from Job Assembly 1 and sent to Job Assembly 2. Access to the first half of the disc takes place over a wide area and in a fairly random fashion, but access to Job Assembly 1 is localized to a small area. Thus Job Assembly 1, unless it exceeds 30,720 words, becomes a "movement-independent" area as described above. This has a significant reducing effect on job organization time.

The compilers

Two compilers were written specially for the Egdon system, a FORTRAN (EGTRAN) compiler and a User-code compiler. Both operate entirely in the core store, i.e. they are one-pass compilers, and both compile one routine at a time. The routines are subsequently read back and bound into a complete program by a system program called the *Relocator*. Relocator also carries out some optimization of FORTRAN routines in the light of extra information which is available at relocate time which was not available at compile time.

The FORTRAN compiler compiles from a dialect of FORTRAN II called EGTRAN which is not appropriate to discuss at length here. All the usual facilities of FORTRAN II are included, together with certain additional facilities peculiar to EGTRAN. In particular the following may be noted:

(1) A facility for varying the dimensions of arrays at run time without sacrificing the optimization of subscripts.
(2) Recursive functions and subroutines, with preservation of the values of variables between recursions if required.
(3) Some additional statements for transferring whole arrays to or from disc or magnetic tape, rather than the normal FORTRAN "list" of variables. This makes it possible to make transfers proceed simultaneously with each other and with central processor calculations, which cannot be done (except to a very limited extent) with "list" type statements because of the need for store protection.

The User-code compiler or assembler accepts a punched card version of ordinary KDF9 User-code. Its output takes the form of relocatable binary routines which are completely interchangeable with those produced by the EGTRAN compiler.

Operational experience

Some seven months operational experience has now been gained on the Egdon system. It has behaved substantially as expected, being an efficient job-shop operating system capable of reducing the time between the submission of a job and the printing of results to a minimum.

The combination of in-core compiler and independent compilation of routines means that large jobs can be amended and made ready for execution in a very short time. For example, a particular multi-segment program comprising 26,000 words of instruction in all (say 8000 EGTRAN statements) takes about 2½ minutes to organize and relocate. This means that this is the time it takes to make a trivial amendment to the program and to start to execute it. This time will be significantly reduced when certain improvements which are being made to reduce the number of disc accesses are completed.

The compiler itself compiles 300 to 350 EGTRAN statements per minute. Object program efficiency is not as high as can be achieved with a multi-pass compiler, but is typically 1·5 to 4 times slower than hand-coded program.

There are some points on which it can now be seen that the system is inconvenient, the most important being pseudo-off-line output. The volume of printed output has been less than was expected, and the facilities for coping with large volumes have therefore been largely

redundant. "Rush-hour" mode is currently not being used at all, production mode with a small print limit being used instead. Modifications will probably be made to simplify the system.

A number of program errors have, of course, been found after the passing of the acceptance tests. The majority of these have been in the Director area, which is not surprising because supervisory programs, being dependent on real-time events, are notoriously difficult to debug. No run has ever failed, however, for a reason that was not fairly easily circumvented, and there has never been a complete stoppage of work because of errors in the software.

Implementation

One of the more unusual features of the Egdon system is that it was completed on time and passed two stringent series of acceptance tests. Since this cannot be entirely explained by the brilliance of those who have worked on it, it is worth while examining other factors which may have contributed.

First, there are political factors. Since the customers for whom the system was written had previously suffered from a late delivery of software from an American manufacturer, a stiff penalty clause was built into the contract. This in itself did not ensure success, but it led to an attitude of urgency on the part of all concerned which was a major contributory factor. For example, it made the customer particularly careful to fulfil his obligations to us, such as providing us with information by a given date, and to avoid too-frequent changes of mind.

Secondly, PERT critical path scheduling was carried out regularly. The writing of software is not an ideal activity for this kind of control because there are so many possible ways of achieving the final object. This meant that the network underwent several major revisions during the course of the project. But in spite of this and of the large amount of unproductive clerical effort involved in preparing data, the existence of the PERT caused a kind of planning-consciousness to be created which had an important bearing on success.

Thirdly, and perhaps most important of all, was the existence of clear objectives from the start. The fact that there was a formal customer-supplier relationship, and that the customer was already an experienced computer user, meant that decisions were arrived at by controlled discussion, and properly documented. This does not mean that every detail was worked out in advance and that nothing was changed—in fact only a general outline existed at first, the details were worked out progressively over a period of several months, and decisions were reversed on several occasions. But the existence of a formal relationship, and the knowledge that a penalty clause existed, damped down the oscillations that this kind of discussion often gives rise to, and meant that a clear definition was arrived at quickly and changes were kept to a minimum.

Implementation from the first planning meeting to the beginning of the acceptance tests, took about 15 months. A total of 20 man-years' programming work was involved altogether, split roughly as follows:

	Man years
EGTRAN and required library	7
User-code Assembler	4
Director	3½
Relocator	1
Disc Update	1
Job Organizer	½
Auxiliary programs	3
	20

Acknowledgements

Dr. L. H. Underhill and Mr. I. C. Pull of A.E.E. Winfrith, and Dr. K. V. Roberts and Mr. L. A. J. Verra of Culham Laboratory played a major part in defining the system and made many valuable suggestions concerning implementation. The following, of English Electric-Leo-Marconi Computers Ltd. unless otherwise stated, helped to implement it:

J. W. Adams, G. M. A. Bernau (Culham Laboratory), D. C. Bindon (A.E.E. Winfrith), T. R. Clayton, J. K. Ebbutt, R. Godfrey, A. J. Harding, J. E. Hartley, P. L. Havard, A. J. Heyes, Miss P. Higson, E. F. Hill, G. H. Johnston, F. D. McIntosh, D. G. Manns, B. F. J. Manly, J. O'Brien, A. C. Peters, A. J. Robbins, A. Sutcliffe, J. G. Walker, P. J. L. Wallis, B. C. Warboys, A. M. Yates.