The MINIMOP multi-access operating system

T. G. McLain* and A. R. Trice†

* *Control Data Corporation, Washington D.C., U.S.A.*
† *Computer Analysts and Programmers Ltd., Reading*

(Both formerly of Software and Applications Programming Organisation, I.C.L., Bracknell)

This paper describes the design and development of a multi-access system which has been implemented on the small to medium 1900 series computers.

(Received August 1969)

Multi-access or Time Sharing systems are usually associated with either large powerful computer installations servicing scores of terminals, or are implemented as dedicated systems on smaller computers using specially designed hardware and software. However, almost all computer users could benefit from some of the facilities typified in such systems, e.g. on-line program debugging, file enquiry, problem solving, etc. It would therefore, seem desirable to make such facilities available on smaller computers, to which communications hardware has been added.

We were, therefore, required to design and implement a simple but flexible multi-access system which would be viable on any 1900 series computer with as little as 16K words of core store, and which would require no special hardware other than that currently available in the 1900 range, and which would interface with the standard 1900 Executive program. This latter requirement was essential, since for processors having sufficient core store, the ability to multiprogram a multi-access operating system with other programs, or indeed with another operating system controlling the scheduling of programs in a batch processing manner, was very desirable. The essential peripheral equipment necessary to operate a simple but efficient multi-access system, consists of communications equipment and some form of random access backing store. Implementation on relatively small processors dictated the use of inexpensive hardware and thus the minimum configuration specified for the MINIMOP system is

(i) Any 1900 series processor (other than a 1901, 1901A or 1902) with at least 16K words of core store.
(ii) Exchangeable Disc Transports, each of 1 or 2 million 24 bit words capacity.
(iii) Communications Multiplexer and Line Terminal Equipment.
(iv) Teletype 33 Consoles.

In addition, it was assumed the processor would have at least one input and one output peripheral (for example, a tape reader and a line printer).

The choice of an Exchangeable Disc store (E.D.S.) in preference to fixed disc or drum was made for reasons of storage flexibility as well as cost. Where the potential users can be grouped in such a way that each group needs access to the computer only during predetermined periods, then it is not necessary to have the total system storage accessible at all times. Exchangeable Disc storage makes this possible and allows each group of users in such an environment more file storage than they would normally have on a fixed disc. This is particularly applicable to University departments.

Facilities provided

Our aim was to provide as many of the basic essentials of a large general multi-access system as space would allow, without including facilities which would increase the complexity of the system to an extent where it was attempting the job of a fully fledged operating system offering on-line and batch processing capability.

These basic essentials were considered to be:

A simple easy to use Command Language.
Adequate filing and editing facilities.
Compilers interfaced at command level, for running programs written in a high level language.
Access to a library of other compilers and programs.
Conversational problem solving capability.
Capability of allowing the user to extend the system under certain limitations to provide the on-line facilities relevant to his application.
Internal batch processing to utilise any unused C.P.U. time if multiprogramming under Executive is not possible (background jobs).
Adequate utility programs to set up the file structure and provide system off-lining.

The first five formed the requirements on which the initial design was based and the need for the sixth became apparent during the development of the free standing 1900 JOSS type conversational language JEAN, which was being undertaken at about the same time. We considered extendability to be the answer in a small multi-access system, to the different needs of a variety of users, and we decided therefore to design the input/output interface between JEAN and MINIMOP in such a way that other conversational packages, written by the user, could be incorporated in the basic MINIMOP system in the same way. For this reason the interface, which is provided by a single monitorable extracode, was made as simple as possible. To have access to such a package it is only necessary to incorporate it in the library with a single marker bit set to indicate its special properties.

The last two requirements were included for completeness. In a 16K processor there would be no unused core space and multiprogramming under Executive would not be possible, so the ability to process background jobs within MINIMOP to absorb unusued C.P.U. time is an added advantage. The provision of utility routines to allow bulk off-line input and output relieves the user of a great deal of the tedium from his work and enables him to obtain required results more quickly. He also needs adequate routines to set up basic file structures, etc. (No attempt to describe these is made in this paper.)

Finally, we attempted to provide as much security and accounting as was possible within the obvious restrictions of a small system. We were unable to provide incremental dumping and automatic archiving facilities during on-line running because of space restrictions. A certain reliance is therefore placed on the organisation of the installation to dump cartridges or files using standard library routines to other cartridges or other magnetic media at regular intervals. File security, in addition to that normally provided by Executive, was achieved by normal password techniques. The system allows the user to change his password at will from the console once he has gained access to his file, thus increasing its security. Accounting information, giving total console connect time and C.P.U. time used, is typed at the console when the user logs out. A record of this is also written to the system output file for central accounting purposes. It was obvious that users with large processors would require more extensive facilities than those available in a system designed for the minimum configuration. For this reason we developed two separate versions, MINIMOP 1 under which users could run programs of up to 5·75K words, and MINIMOP 2 under which users could run programs of 11K words. Because of the core size limitations, MINIMOP 1 was only capable of providing JEAN and a FORTRAN II compiler, however MINIMOP 2, as well as providing JEAN, has integrated ALGOL, FORTRAN IV and PLAN 3 compilers. These compilers are standard ICL compilers interfaced directly to MINIMOP. (The Command Language is given in Appendix 1.)

File store

It was our objective to provide a simple but adequate filing system for users. The 1900 Executive program contains a file control package for E.D.S. and there seemed no reason not to take advantage of this and avoid duplication within MINIMOP. We therefore arranged that a user at a console could have access to one named file on disc at any given time. The name of the file, as known to Executive, is synonymous with the name of the user, as quoted in the logging-in command, and there is a one to one correspondence, therefore, between console and file. (Subsequently we discovered this to be an annoying restriction when considering enlarging the system.)

Blocks of information are stored within a given user's file in named subfiles and a directory of these subfiles is maintained in the first few blocks of the file. One ever-present subfile is named SPACE and records the amount of unused storage at the end of the file. File handling at the subfile level is performed entirely by MINIMOP. Subfiles are flagged according to the type of information contained in them, and in particular erased subfiles, which contain information no longer required, are flagged in a unique way. All such erased subfiles are physically removed from the file and the remainder arranged sequentially in the file, with the SPACE subfile adjusted in length, after condensing the file (garbage collection). This can occur as a result of a command from the user, or by system instigation when a command implicitly requires more disc storage than there is available at the end of the file.

In addition to users' files, of which there could be any number within reason on one or more E.D.S. cartridges, several system files were essential, namely

A file to contain a copy of MINIMOP and its overlays (MINIMOP file).
A dump file to contain the current state of each program associated with a particular console when not resident in core (the SWAP file).
A file for off-line output and accounting information (the SYSTEM OUTPUT file).
A file to contain a prepared sequence of programs and their data to be run when the system is underloaded (the BACKGROUND JOBS file).
Several library program and subroutine files.

All system files are administered by Executive in the same way as users' files. The SWAP file contains a program loader in the first block to enable library programs to be loaded to core initially on command from a user. It is situated here for convenience since it is easy to bootstrap it in using the program swapping sequence.

Basic system design

The structure of the system reflects the three basic functions which have to be carried out, namely:

Input/output functions
File referencing functions
System activity and user program scheduling

In order that these functions could be carried out apparently autonomously, we took advantage of the sub-programming facilities provided by Executive in the majority of 1900 processors. This is a device whereby different parts of a single program may be multiprogrammed with each other, by providing a copy of the basic registers with each subprogram (member). Thus we arranged that MINIMOP would consist of three members each concerned with one of the functions listed. Communication between each member program is by common queues and lists which are formed, maintained and processed by each of the member programs (see **Fig. 1**).

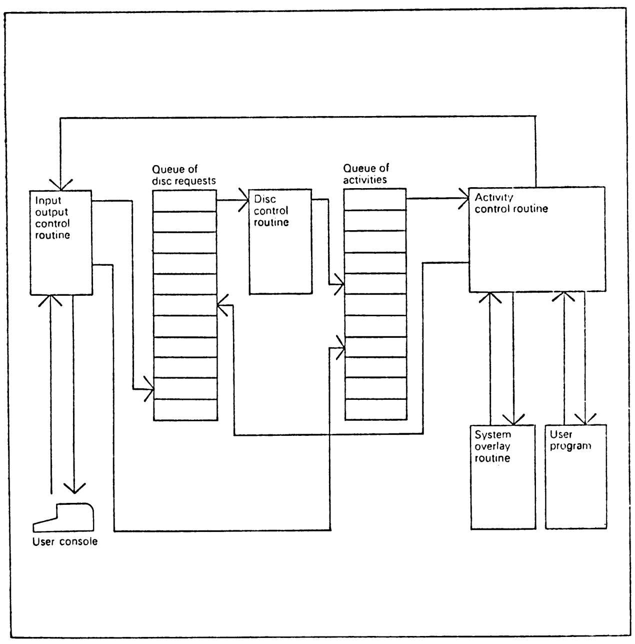

Fig. 1. Basic system design

The provision of buffers presented a somewhat difficult problem, remembering that we had severe core storage limitations. We arranged for a single dedicated buffer to be assigned to each console within the program, capable of containing 128 characters in internal 1900 code. This was used both for input from and output to the console.

For file transfers, a number of 128 word buffers are provided, and allocated and relinquished dynamically (four such buffers are currently incorporated in MINIMOP 2). In spite of this minimal bufferage, the core store occupied by buffers occupies nearly 1K words, approximately one third of the non-overlayed part of MINIMOP.

In addition to these buffers, for each console, an area of working store was set aside, called the V-area. It contains various values associated with the state of the particular console and any activities being carried out for the console user by MINIMOP. For example, the location VO contains indicators specifying the physical state of the console, e.g. the multiplexer channel number to which it is attached, whether it is inputting or outputting, etc. This area of store is referred to by individual routines of MINIMOP using a modifier and quoting V*n* in the address field of a typical instruction. This enables the majority of these routines to be multithreaded.

A particular use which the V-area is put to is to store the current peripheral assignments of a console user. Quite often the program the user is running from his console expects input from, and sends output to ordinary peripheral devices. However, the user usually requires to input directly from his file and output either to his console or to the system output file. In order to do this, he 'assigns' each such peripheral referred to in his program, to be effectively a named subfile in his file, the console or the system output file.

There are three basically separate program instruction areas in core store:

An area for routines fixed permanently in core store.
A 256 word overlay area for system activity routines.
An object program area into which the current object program is swapped when allocated processor time.

Having these three areas gave us considerable flexibility during the development of the system, since any particular routine, if it cost too much in fixed store, could be overlayed as a system activity routine; and subsequently, if it was necessary to enhance it so that it occupied so many overlays that its efficiency was impaired, it could be converted to what we termed a subject program and be loaded to the object program area from the library. The Editor is an example of such a subject program competing with any other user's object program for processor time, whereas the LOGIN system activity is performed by several system overlays at a higher priority than subject programs.

Input/Output

The fundamental approach to input/output is that the user operates in a question and answer mode, thus, unless the system is outputting to a console, it is always in a state of readiness to accept input, and will have indicated this to a user by printing an 'invitation to type' character. An exception to this rule is when a user enters a lengthy process from his console (e.g. a compilation), in which case he is informed that the process has started but of course his console then remains temporarily dormant.

Four levels were defined for the input mode, as follows:

Level 0	Prior to logging in
Level 1	Command level (post logging in)
Level 2	Conversational level (input direct to program)
Level 3	Data to file input

At Level 1, all MINIMOP commands except LOGIN are valid. Certain commands lead to level 2 being set, whereby all input messages are passed on by the system to the user's object program. This could be, for example, the MINIMOP Editor which would interpret such messages as editing commands, or JEAN, which would interpret such messages as JEAN commands; Level 3 is set on receiving an INPUT or MACDEF command at Level 1, and all subsequent input messages, until the terminator **** is input, are automatically written to the user's file.

File referencing

All transfers to and from any user's file, except the library files and the swap file, are queued chronologically in a circular list by entering a common subroutine. It is a feature of the system design that only one transfer (except in exceptional circumstances) can be queued for a given console at any time, so the list is of finite length. Each transfer request queued, in addition to the transfer parameters, specifies whether a buffer is required, whether on completion a standard MINIMOP subroutine should be entered, and whether the activity which is temporarily halted, awaiting the completion of this transfer, is to be requeued in the list of current activities. Transfers may be one of three types, read, write, and read-then-write. The subroutine call feature is especially useful in the case of the last type of transfer because a subroutine can be entered automatically at the end of the read phase to move information selectively into the buffer prior to the write phase, thus achieving selective record overwriting in the file. The subroutine can also change itself to another subroutine so that a second subroutine can be entered after the write phase, which we found a useful trick.

Activity and program scheduling

In scheduling the available processor time between different consoles, reference is made to an activity list, which has one entry per console. Every MINIMOP command results in a chain of activities being carried out, some of which may be system activities, some of which may be program activities. System activities are carried out exclusively by MINIMOP overlay routines

```
MINIMOP 1 IS READY

LOGIN ROBINSONFILE
TYPE PASSWORD
←××××I                                          User types password on illegible 4 characters
OK
←SUMMARY                                        User obtains details of the contents of his file
S P A C E      255      144
H H U U F      242        8
T T D A          8        1
D B T S M        9        9
P P L P         18        6
MMMMF           24        9
MB MB F         33       27
J KJ KF         60        9
D B B C M       69       36
L D A D F      105       12
MNB U S        117       53
I  T P R       170       24
L L L L F      194        8
L L L MF       202        8
P C1 F         210        8
P C2 F         218       35
T T T U        253        1
T T T I        254        1
OK
←LIST TTDA                                      User obtains details of the contents of a data subfile
3000            0
1
2 3 0
  4500 7125
  50
3000
  0
OK
←EDIT TTDA                                      User calls in Editor to edit this data subfile
OK
←WC = /                                         Defines warning character to be recognised by the Editor
OK
←/ALT 0, 1, 300                                 Alters first line to read 3000 500
OK
←3000 500
←/FINISH                                        Exit from Editor
OK
←FORTRAN ITPR, ZZZJ, CONS                       Compile FORTRAN program held in subfile ITPR into subfile
OP                                                  ZZZJ sending listing to the console
←FORTRAN COMPILATION BY # XFDE MK IC DATE
     20/11/69
C        PROGRAM FOR CALCULATING INCOME TAX
C        PROGRAM DESCRIPTION
         PROGRAM (PAYE)
         INPUT 1 = TRO
         OUTPUT (MONITOR), 2, 3 = LPO
         END
C        ONLY ONE SEGMENT
         MASTER PAYE
END OF SEGMENT, LENGTH 186, NAME PAYE
         FINISH
END OF COMPILATION—0 ERRORS
DELETED EP
←LOAD ZZZJ                                      Load program
OP
HALTED LD
←ASSIGN LPO, CONS                               Specify LP output to console and TR input to come from subfile
OK                                                  ITDA
←ASSIGN TRO, ITDA
OK
←ENTER 0                                        Execute program
TOTAL TAX PAID FOR THE YEAR = £547·76
DELETED
←LOGOUT
```

Fig. 2. Example of MINIMOP console log

and have an inherently higher priority than program activities. The latter need much more core store, and progress for between one and two seconds until they are complete or until they require MINIMOP to perform some service, whichever is the sooner. Double round robin scheduling is applied, in that all currently queued system activities are serviced one after the other between the servicing of each program activity. In practice, however, since nearly all system activities require the overlay area, what happens is that the overlay for the first system activity is transferred to core store while the first program activity is progressing. When this terminates, the system activity is obeyed if the overlay is in core and when complete, the transfer of the overlay for the next system activity is started and the next program activity progressed. Naturally there are variations in this pattern, depending on the mix of system and program activities, and the occurrence of consecutive system activities using the same overlay. However, a fair amount of overlap is achieved by this double-round-robin scheduling algorithm and considerably more could be achieved with the introduction of more overlay areas and more than one program area; unfortunately we were unable to spare the space.

Object programs are given at least one second as mentioned above, however MINIMOP monitors the running of the object program and, if a service is required, it may be either swapped out and placed at the bottom of the program activity queue, or suspended while the service is carried out. An example of the latter is the occurrence of an input instruction referencing a basic peripheral which has been assigned as a subfile of a user's file. An example of the former is an output instruction in conversational mode to the user's console. Other events in the object program such as illegals, program messages to the operator console, program deletions, etc., are dealt with similarly.

System development

The total suite of programs was written in the 1900 PLAN assembly language using the 1900 COSY editing and assembly system. One of the difficulties encountered in testing even a small multi-access system, catering for only a few user consoles, is the inability to feed the system with pre-arranged sequences of input from a selection of terminals. To simplify this problem a communications simulator was included in MINIMOP. This enabled MINIMOP to read simulated commands for several imaginary terminals from paper tape and reflect them and their responses to a line printer. Thus we were able to feed MINIMOP, when under test, with predetermined sequences of commands for several lines to eliminate particular errors. Because of the higher speed of this paper tape reader/line printer combination than that of a number of teletypes, we were able to carry out tests more quickly and, what is more important, remotely, since these tests needed no interaction on our part. This testing technique is extremely useful for the bulk of the development of a multi-access system and solves the problem of finding numbers of tame teletype operators when testing. However, saturation testing, with real consoles interacting with the system, was of course essential to eliminate errors not detected using the communications simulator, for example timing errors.

System performance

MINIMOP 2 totals about 10K words of permanent and overlayed on-line program, about 5K words of subject programs and 14K words of utility programs. It is now in use in about twenty installations, in the majority sharing the core with one or possibly two GEORGE 2 batch operating systems. Our general impression is that, once the initial system set up procedure has been accomplished and the users of the system have become used to it, they find it extremely easy to use and, because of this simplicity of use, find that they get through their program development and problem solving much more quickly than they would by other methods. In addition it is proving a useful tool in University installations where students are undergoing courses in computing.

Nevertheless, from experience, we appreciate that there are several areas in which we would have liked to have done things differently. However, MINIMOP is being continually enhanced to include additional facilities which have been found desirable by users of the system.

International Computing Services Ltd. have developed from MINIMOP 2 the commercial multi-access system INTERACT, which operates on a 64K 1905F. This system, currently allowing nine simultaneous users, will be developed and enlarged in the future to allow many more users.

Acknowledgements

We would like to thank ICL for permission to publish this paper. We would also like to thank the original team of five programmers whose unenviable task it was to do the programming and testing of the system and to Mr. M. Edmunds of ICSL for his many constructive suggestions.

Reference

MINIMOP manual. ICL publication 4102

Appendix 1

MINIMOP 2 COMMAND LANGUAGE

ALGLOAD	sfn, CONS*	Compiles and loads the ALGOL program in the subfile.
ALGOL	sfn, sfn, CONS*	Compiles the ALGOL program stored in the first subfile and writes the semi-compiled program to the second subfile.

ALTER	m, n	Resets the contents of location m in the users program with value n, m and n have several formats.
ASSIGN	peripheral name, sfn	Input assignment.
ASSIGN	peripheral name	Output assignment.
ASSIGN	peripheral name, CONS	Output assignment.
CONDENSE		Removes all deleted subfiles from the users file.
DATE		Prints the date.
EDIT	sfn, sfn*	Loads the Editor in readiness to edit a subfile.
ENTER n		Enters the users program at location 20 + n.
ERASE	sfn	Marks this subfile deleted in the file.
FIND	program name	Loads the specified program from the library.
FORLOAD	sfn, CONS*	As ALGLOAD but for a FORTRAN program.
FORTRAN	sfn, sfn, CONS*	As ALGOL but for a FORTRAN program.
INPUT	sfn	Introduces a new subfile. Subsequent lines of input up to **** are stored in the subfile.
JEAN		Loads the JEAN program and enters it.
LISTFILE	sfn, peripheral name*	Output the subfile to the console or system output file.
LOAD	sfn, CONS*	Consolidates and loads the semi-compiled program in the subfile.
LOGIN	file name	Opens the users file and prepares to check the password.
LOGOUT		Disconnects the user from the system closing his file and printing accounting information.
MACDEF	sfn	As for INPUT but each line of input represents a command.
NEWPASSWORD	password	Changes the password of the users file.
OBEY	sfn, X, Y, Z . . .	Implements the commands listed in the subfile. X, Y, Z, etc., are actual parameters which replace original undefined parameters %A, %B, %C, etc.
PLAN	sfn, sfn, CONS*	As ALGOL but for a PLAN program.
PLNLOAD	sfn, CONS*	As ALGLOAD but for a PLAN program.
PRINT	m, n	Outputs the contents of locations m to m + n − 1.
RESTORE	sfn	Restores the users saved object program as his current object program.
SAVE	sfn	Saves the user's current object program in his file.
SUMMARY		Prints a summary of the users file at the console.
TIME		Prints the time.

N.B. * denotes optional parameter, sfn denotes subfile name

Book Review

Numerical Methods for Partial Differential Equations, by William F. Ames, 1969; 291 pages. (*Thomas Nelson and Sons Ltd.*, £3.25 paper.

In the preface, the author states that the present volume constitutes an attempt to introduce to upper level engineering and science undergraduate and graduate students the concepts of modern numerical analyses as they apply to partial differential equations. In his attempt the author has most certainly succeeded and has produced a most readable and lively text. The contents cover both initial, boundary and eigenvalue problems with reference (where appropriate) to elliptic, parabolic and both first and second order hyperbolic problems. The author's style of giving reasonably meaningful practical examples of application of the methods combines with his semi-rigorous approach to the more theoretical problems of stability and convergence.

The student will find some 250 exercises spread throughout its five chapters whilst 400 references (up to the year 1968) will be of more interest to the more experienced reader. If I were to find fault with this book it would be in the scant treatment given to multidimensional problems in comparison with intensive discussion of one and two dimensional problems. However, the author is probably justified in the virtual exclusion of such topics as his aim is primarily to produce a student textbook. With this in mind the author is to be congratulated on the outcome.

A. R. Gourlay (Dundee)