Job control in the MU5 operating system

G. R. Frank

Department of Computer Science, University of Manchester, Manchester M13 9PL

This paper describes the job control part of the MU5 operating system. The operating system provides a set of virtual machines for the execution of user jobs and operating system tasks. Each has the ability to create and control other virtual machines which run in parallel with it. A number of special processes called supervisors exist within the system solely to perform job control functions up to the point at which a new virtual machine is created. After this point the user process, running in the new virtual machine, performs most job control functions for itself. The design is such that job control requirements may be expressed in any programming language supported by the system, so that no job control language is actually necessary.

(Received September 1974)

This paper presents the main features of a job control system which has been designed and constructed for the machines of the MU5 complex at Manchester University. The system is applicable to a wide range of machines, and is designed primarily to simplify the task of making major or minor modifications to the user interface. Morris (NCC, 1974) has already discussed the MU5 job control facilities in terms of their development from other job control systems, particularly Atlas. In this paper the MU5 system itself is discussed in greater detail.

1. The MU5 operating system

The MU5 complex consists of a number of interconnected computers. Each is considered as an independent machine and can, if required, run a complete and independent operating system. In the normal mode of operation, however, operating system modules are distributed at any given time among the machines which are currently operational. Within each machine in the complex there is a small operating system kernel whose function is to map a set of compatible virtual machines on to the actual hardware resources. It is within these virtual machines that the remaining operating system functions and user jobs are performed, each running as a separate process in its own virtual machine (Morris, Detlefsen, Frank and Sweeney, 1971).

Each virtual machine has a segmented virtual store into which a number of 'common' segments are pre-loaded. The common segments are write-protected from the user, and contain an extensive set of library procedures available automatically to every process. In addition to the mathematical functions and input/output procedures required by most programming languages, the library also includes all of the compilers, editors, and similar programs. This library forms the basis of the job control system described in this paper.

Certain of the procedures in the library interface with the operating system kernel. These provide the process with an ability to manipulate its own virtual machine, and also to create, control and communicate with processes in other virtual machines. As with the other library procedures, these may be called by any process.

Interprocess communication (and therefore communication between operating system modules) is by means of a message passing mechanism which enables any process to send messages to any other. The message system extends over all the machines in the complex, so that system modules can easily be moved from one machine to another without affecting other modules. A full description of the message system may be found in Morris, Frank and Sweeney (1972).

The input/output facilities of the virtual machine are entirely based upon the message system. All input to a process arrives in the form of messages from other processes; all output is sent as messages to other processes. Thus the concept of a peripheral does not exist within the virtual machine. Instead, actual physical input/output is dealt with by a number of special system processes called *device controllers* which are able to drive the peripherals directly. These are responsible for converting between physical input/output documents and messages. This is their sole function. Thus for example an input device controller is simply a means whereby the user can inject messages into the system. Input may be directed at any process by specifying on the first line of any input the name of the process to which it is to be sent.

2. The job control supervisors

Clearly if the device controllers are to perform only message switching functions there must be other processes in the system to which input can be directed. The particular process chosen will depend on what the user wishes to do. For example it may be an output device controller if he simply wishes to obtain a listing of the document, or a file manager process if he wishes to file it, or it might be a user process. If the user wishes to start a job, however, he will normally address his request to a process which resides in the system for this purpose. Such a process is called a *supervisor*.

The general philosophy in the design of the job control system has been to arrange that the process created to execute a user job, rather than its supervisor, be responsible for the majority of job control functions. This is possible because of the existence within every virtual machine of the full set of library procedures; most of the operations commonly dealt with in a job control language simply correspond to library procedures in MU5. The task of a supervisor is thus kept small, and the supervisors are small, manageable modules. Also, the isolation of most job control functions within the user virtual machine means that an error in the implementation of these functions only affects a single job, whereas an error in a supervisor might affect many jobs.

Obviously the most important function of a supervisor must be actually to initiate the process in which a user job is to run. It does this simply by calling library procedures which interface with the system nucleus, specifying the name and password of the user to be charged for the process. No privilege is required to call these procedures, so that in principle any process may act as a supervisor. Thus new supervisors can be added at any time and a user who wishes to do so may provide his own supervisor. However, it will usually be more convenient to use the standard system supervisors.

One major job control function which is not easily dealt with within the user virtual machine is scheduling. There are two main objectives of scheduling in a computer system. The first is concerned with obtaining efficient use of the machine. Although this may involve some decisions at the job control level (for example to optimise the mix of jobs actually presented to the kernel) it is primarily the concern of the kernel. The second objective, however, is concerned with achieving specified response and turnaround times for the users of the system. This is clearly the concern of the job control system.

In MU5, a supervisor is not expected to interfere at all in the detailed scheduling of user processes. Instead the supervisor is able to specify a priority level for each process it creates. The priority level affects the way in which CPU time is allocated to a process by the kernel. Use of the higher priority levels is limited by relating the charge for CPU time to the priority level at which a process runs. The same priority mechanism is used to determine the relative priorities of the supervisors themselves and of other system processes.

The system design deliberately allows for the simultaneous existence of several supervisors. The main reason for multiple supervisor operation is to enable the diverse needs of different classes of users to be handled efficiently without the need to construct a large, all powerful supervisor. Another important advantage, however, is the ability to create and test a development version of a supervisor without affecting users working with the original version. The input to a supervisor simply consists of messages, and may originate from an input device controller or be created as the output from any other process.

3. Job control within the virtual machine

The main job control functions which remain to be performed within the user virtual machine are concerned with providing a means of

1. Performing some initialisation of the virtual machine in which the job is to execute, for example by assigning actual files and input/output documents to the logical input/output streams of the process.
2. Specifying the sequencing of subtasks within a job.
3. Handling contingencies which arise during execution, especially error conditions.

These operations form the bulk of job control statements in most systems.

In MU5, both initialisation of the virtual machine and sequencing of subtasks are achieved simply by calling appropriate library procedures. For each facility in the virtual machine which requires initialisation, a library procedure exists to perform that initialisation. For example by calling the library procedure

SET FILE INPUT (2, 'FILENAME')

a process can assign the file 'FILENAME' to its logical input stream 2. Similarly the sequencing of subtasks is achieved by calling library procedures, as all the compilers, editors and other such programs which commonly form subtasks within a job exist as procedures in the system library. Thus for example the ALGOL compiler is entered simply by calling the library procedure ALGOL. It will return control at the end of compilation to the calling program which can then arrange, again by calling library procedures, to enter the compiled program or define it on a file for later execution.

Since all job control steps consist of calls to library procedures, the main requirement, to be able to handle errors and other contingencies, is a means of communicating status information from library procedures. Two mechanisms are used in the MU5 library. Firstly, each library procedure sets a global 'status return' parameter before returning, indicating whether or not it successfully completed its task. This status can then be tested by the caller. Secondly, certain types of serious error condition cause an interrupt within the virtual machine. The effect of this is that control is forced into a trap procedure, which may be either a procedure supplied by the user or a standard system monitoring procedure. These two mechanisms may be used by a job control language interpreter to provide any required degree of control.

4. MU5 job control languages

As a consequence of the system structure outlined above, the 'job control language' seen by a user of the MU5 system will usually consist of instructions to three different processes, though this need not be apparent to the simple user of the system. The three processes are

1. The input device controller
2. The supervisor
3. The process executing the job

The instructions required by the device controller are minimal, serving only to delimit the start and end of each input document and identify the supervisor to be used. **Fig. 1** shows the device controller commands for a batch document and an online communication.

With most supervisors the instructions to the supervisor will also be small, serving mainly to supply parameters to the 'CREATE PROCESS' library procedure (time limits, size limits, etc. for the virtual machine). In most cases these instructions can be placed on the same line as the device controller commands. Thus the instructions to the process running the job form the bulk of the job control language.

As described in the preceding section, a job control language for MU5 requires a means of steering the process through a series of library procedure calls and providing the correct parameters. Control statements and a means of testing the status returned from library procedures are also probably desirable. Barron and Jackson (1972) have observed that job control is a form of programming, and that existing job control languages are similar to assembly languages or simple autocodes. In MU5 the fact that the main operations are procedure calls makes the use of a programming language seem even more natural.

The MU5 library is organised with a fully-recursive ALGOL-like procedure structure, to the extent that even the compilers may call one another if required. Also the parameter specifications have been chosen such that, although the treatment of parameters will be language dependent, it should be possible to generate all parameter types (and thus call all library procedures) from a program in any programming language. This is achieved by specifying parameter types for library procedures in terms of their physical properties (e.g. size) leaving their interpretation in terms of logical entities to individual compilers. The parameter types allowed for library procedures are:

1. A single length (32-bit) operand, usually an integer
2. A double-length (64-bit) operand, which may be either a real number or a long integer
3. A descriptor, yielding a vector or string

(a) Batch input document

***A PROCESS-NAME USER PASSWORD

(The document itself, sent as a single message to the named process when the terminator is found)

***Z

(b) Online communication

***M PROCESS-NAME USER PASSWORD

(Lines of input, sent a line at a time as they are typed to the process named)

***Z

Fig. 1 Device controller commands

An ALGOL compiler might allow the following substitutions for these parameter types:

1. An integer expression
2. (*a*) A real expression
   (*b*) A string of not more than eight symbols which will be packed according to the system convention for names (user names, process names, file names, etc.).
3. (*a*) A symbol string, for which a descriptor is constructed
   (*b*) A vector name
   (*c*) A variable name, yielding a descriptor to the variable.

Similar interpretations can be allowed in other high level languages, so that each compiler can make all of the library procedures available to its users. Indeed the library procedures appear as pre-declared procedures in the target language into which the MU5 compilers translate (Capon, Morris, Rohl and Wilson, 1971). This means that any programming language available on MU5 can be used for job control purposes.

This approach makes available at the job control level the full power of a programming language. Furthermore, the user is able to use the programming language with which he is most familiar, rather than having to learn a completely new language for the purpose. As an example of the use of a programming language for job control, see **Fig. 2.** This is a program in ALGOL to run student ALGOL programs against standard test data, and mark the results. Its input consists of a batch of student jobs on the file named 'programs', and the test data on the file named 'testdata'. Each job is assumed to begin with the student's name on a line by itself, followed by the program, and the student's own data terminated by the symbol $. Two output streams are generated, output 0 being a list of student names and their corresponding marks and output 1 the compiler monitoring and actual results from each job. Only jobs which compile successfully are entered. The *procedure* which does the marking is not given, but it is assumed that it somehow compares the output generated with some standard results and assigns and prints a mark.

Examination of actual jobs shows that the great majority of jobs are not nearly so complicated, and do not require the power of a programming language. Most jobs are very simple at the job control level, and for these it may actually be inconvenient and would almost certainly involve significant overheads if a full programming language were used. A simpler 'job control

```
***A JOB USER PASSWORD TITLE
ALGOL ()
(. . . ALGOL PROGRAM . . .)
ENTER ()
(. . . DATA . . .)
***Z
```

Fig. 3 Simple ALGOL compile-and-go job

```
***A JOB USER PASSWORD ECOMP
SET FILE IO (1, 1, 'SOURCE')
EDIT (1, 1)
(. . . the editing commands . . .)
END INPUT(1)
END OUTPUT(1)
SET FILE INPUT (1, 'SOURCE')
ALGOL ()
**SELECT INPUT (1)
ENTER ()
STOP (0)
***Z
```

Fig. 4 Edit, compile and execute job

```
begin integer start, status, ch;
  set output (1, ⟨LPT⟩, 10000);
  set file input (1, ⟨programs⟩)        ; comment programs for testing;
  set file input (2, ⟨testdata⟩)        ; comment standard test data;
  select input (2)                      ; comment remember start position;
  start := i pos                        ; comment in test data;
  select input (1);
  select output (0);
  for ch := next sym while ch ≠ 4 do
    begin                               ; comment next student;
    for ch := in sym while ch ≠ 10
      do outsym (ch)                    ; comment print student name;
    select output (1)                   ; comment for compiler monitoring;
    algol                               ; comment enter compiler;
    status := pw0                       ; comment pw0 globally declared;
    for ch := in sym while ch ≠ ⟨$⟩ do  ; comment discard old data;
    if status ≠ 0 then begin
      select output (0);
      write text (⟨failed to compile⟩) end
    else begin
      select input (2)                  ; comment select test data;
      set i pos (start)                 ; comment reset to start;
      select output (1);
      enter;                            ; comment enter compiled program;
      select output (0);
      mark                              ; comment enter marking program;
      select input (1)
      end
    end;
  write text (⟨end of run⟩);
  step (0)
end
```

Fig. 2 Job control in ALGOL Note: the character code 4 is used to indicate 'end of file' and 10 is 'newline'

language' has therefore been introduced to the MU5 system to satisfy the majority of 'simple' users. All processes begin execution in a common library procedure called START. This creates an output stream zero, and begins to read from input stream zero (which is usually supplied by the supervisor by forwarding the input supplied originally by the user). Successive lines of input are read and executed interpretively as calls to library procedures with literal parameters. On return from each procedure the status returned is examined; if a fault is indicated the process is forced into a trap procedure, otherwise the next line is dealt with in the same way. **Fig. 3** shows a simple job, which compiles and then immediately executes an ALGOL program. **Fig. 4** shows a rather more complicated job, which first edits a source file, then compiles it and finally executes it.

The interpretation of these simple job statements involves little overhead. All that is required is to read the procedure name and its parameters, using procedures such as 'read symbol', 'read character string', 'read integer', etc. and to call the specified procedure. The full power of the library is available, and because of the direct correspondence between job control statements and library procedures, any new facilities in the library automatically become available at the job control level. Sequencing of job steps is specified implicitly: the steps are always obeyed sequentially unless a fault occurs, in which case the process usually terminates. Job control statements can also be inserted in the text of a program being compiled by preceding them with '**'. In this case the compiler detects the '**' and calls the library interpretation procedure recursively. An example of the use of this facility to switch to a new input stream during compilation is shown in Fig. 4.

It has been found that few users actually ever need more than the simple interpretive facility for their job control. Indeed there are many useful jobs which can be performed without ever leaving the 'job control' level. The ability to use a programming language for job control is there when it is needed, without actually costing anything if it is not used.

5. Supervisors in the MU5 system

The present MU5 job control system has evolved as a result of experience with using a prototype system implemented on a modified ICL 1905E which now forms part of the MU5 complex. In the original prototype system it was felt that three supervisors would be required for normal computing service applications, with a number of others fulfilling more specialised requirements. The three standard supervisors were called BATCH, JOB, and ONLINE.

BATCH was a supervisor designed to cater for the large, batch-processing type of job, and provided facilities for the user to specify scheduling requirements (deadlines, etc.), complex input/output requirements and job sequencing. The input/output specifications were similar to those of the ATLAS supervisor (Howarth, Payne and Sumner, 1961) with extensions to deal with files and multiple-document input streams. Job sequencing was based on the notion of 'program events' similar in concept to those of the GEORGE 3 system (Oestreicher, Bailey and Strauss, 1968) and implemented by sending messages from the user process to its supervisor. There seemed at the time to be a strong case for providing another supervisor to deal with the simple jobs which are extremely common in a university environment. This was the JOB supervisor, which provided only minimal facilities and as a result was simple to use and had low overheads. Finally, the ONLINE supervisor handled interactive applications. It was felt that the main requirement here was to allow the user to 'become' the supervisor, by providing him with the means of performing the operations normally performed by supervisors.

As the role of the library developed many of the functions of the BATCH supervisor became redundant, as the same effects could be achieved via the library. BATCH was therefore abandoned for a slightly improved form of the JOB supervisor, which is now the standard system supervisor. The examples in the preceding section show the use of the JOB supervisor. The title line can also specify options such as time and size limits and priority ratings for the job, but the default settings of these parameters are usually adequate. It is still felt that a special supervisor is useful for interactive work (though the JOB supervisor can handle interactive jobs), but little experience has been gained as yet with this mode of operation owing to shortage of online devices.

Supervisors can also exist for special purpose environments where the standard facilities would be inconvenient. Particular cases of these are a special editing system which facilitates editing by relatively unskilled staff, and online transaction-oriented systems such as desk calculator systems and reservation systems. Finally it is expected that a number of supervisors will exist to simulate the job control facilities of other operating systems on MU5.

6. Implementation of other job control languages

In MU5, job control requirements are normally specified by using the simple interpretive scheme already described or by using a programming language. However, this does not prevent the existence of special job control language interpreters in the MU5 system. Indeed it is one of the aims of the MU5 system design that the construction of alternative user interfaces should be quite simple. The main uses of such special interfaces will be to provide facilities comparable with those of other systems, so that for example a user of GEORGE 3 who occasionally wishes to use the MU5 system can do so using the job control statements with which he is familiar.

As a general strategy, the implementation of a new job control subsystem is likely to involve the insertion of a new supervisor and some new library procedures. Either of these may be omitted in particular cases, but the supervisor will usually be concerned with creation and co-ordination of processes while the library procedures map the 'foreign' virtual machine facilities on to the MU5 virtual machine.

It is proposed that in the future a number of 'foreign' job control systems be implemented on MU5 to assess the cost of implementing and running such systems. As a pilot study for this project, an implementation of the ATLAS job control system (Howarth *et al.*, 1961) has already been attempted. The ATLAS system was selected mainly because of its simplicity, which made it suitable for a short investigation. It also has the advantage that the input/output facilities of MU5 are similar to those on ATLAS (i.e. based on numbered logical input/output streams accessed by operations such as read character and print character). This means that an ATLAS job description can relate sensibly to many of the jobs run on MU5.

Because of the similarities in the ATLAS and MU5 virtual machines, it was not necessary to provide special library facilities for use by 'ATLAS' jobs. The ATLAS job description is read by a special supervisor (called ATLAS) which translates into appropriate library calls the statements which in MU5 require action in the user's process. This was felt to be reasonable as the supervisor must in any case scan the whole of the job description to extract the information needed to create the process. Apart from this the supervisor's main functions are queuing of jobs and association of input documents with jobs. (This latter facility is not provided in the standard MU5 system. Jobs which require many input documents are expected to use files for all except the document containing the job control commands.)

As far as the user is concerned, ATLAS jobs may be run directly on the MU5 and 1905E systems, subject of course to the existence of a suitable compiler. The only change needed to the JOB and DATA document is to precede each by a

'***A ATLAS' line to direct them to the appropriate supervisor. No provision has been made for the use of files, but a straightforward extension of the INPUT and OUTPUT sections of the job description would remedy this.

The supervisor was written in MU5 Autocode, as a compiler for this language exists on both the MU5 and 1900 systems. The whole exercise, including testing, took one day and the supervisor was tested and introduced into the system as a normal user program, without the need to interrupt the normal computing service. The supervisor is quite small (less than 1,000 orders) though by no means optimally coded, and its use does not involve a large overhead.

7. Conclusions

The system described in this paper has evolved as a result of experience with the operating system. It has now been in use in almost the present form on the 1905E for about 2½ years and has proved a useable system even though the library implementation on the 1905 is not fully recursive. This latter restriction means that use of a programming language for job control requires some care to avoid overwriting statically allocated data areas. The simple interpretive system has proved efficient and quite adequate for most jobs. Although the ability to use a programming language for job control does exist it is rarely used, and its existence imposes no cost either in complexity or overheads on the user who chooses not to use it.

Since the operating system first became operational early in 1970, the user interface has undergone a continuous process of evolution. Most of the changes take the form of new facilities within the virtual machine and are achieved by additions or modifications to the system library. Such modifications present little difficulty; the library is the only part of the system which needs to be re-compiled for the new facility to become available in all programming languages and in the simple interpretive job control system. Modifications to the supervisors have also been made quite frequently, usually to investigate the usefulness of some new facility. These changes are also quite easy, involving only a recompilation of the relevant supervisor, and can often be made while the system is actually running a service. Certain types of new facility, apart from job control, have also been introduced by creating new supervisors. Finally, the experiment with implementing the ATLAS supervisor suggests that the system can quite easily be made to accept completely different job control languages, either instead of or in parallel with the normal system. However, further research is still necessary to determine how useful this facility can be made. A job control language, by its very nature, is likely to involve some assumptions about the underlying virtual machine structure, and the job control language itself might not be meaningful in the context of a different operating system.

References

Barron, D. W., and Jackson, I. R. (1972). The Evolution of Job Control Languages, *Software—Practice and Experience*, Vol. 2, No. 2, pp. 143-165.

Capon, P. C., Morris, D., Rohl, J. S., and Wilson, I. R. (1971). The MU5 Compiler Target Language and Autocode, *The Computer Journal*, Vol. 15, No. 2, pp. 109-112.

Kilburn, .T. Howarth, D. J., Payne, R. B., and Sumner, F. H. (1961). The Manchester University Atlas Operating System, *The Computer Journal*, Vol. 4, No. 3, pp. 222-229.

Morris, D. (1974). Job Control in Atlas and MU5, Presented at a BCS symposium on Job Control Languages. Proceedings published by the NCC.

Morris, D., Detlefsen, G. D., Frank, G. R., and Sweeney, T. J. (1971). The Structure of the MU5 Operating System, *The Computer Journal*, Vol. 15, No. 2, pp. 113-116.

Morris, D., Frank, G. R., and Sweeney, T. J. (1972). Communication in a Multi-Computer System, Proc. IERE Conference on *Computers, Systems and Technology*, October 1972, pp. 405-413.

Oestreicher, M. D., Bailey, M. J., and Strauss, J. I. (1969). GEORGE 3—A General Purpose Time Sharing and Operating System, *ACM*, Vol. 10, No. 11, pp. 685-693.