The many faces of Multics

J. E. Jarvis

Honeywell Information Systems Ltd., Honeywell House, Great West Road, Brentford TW8 9DH

Figure 1 is an example of a standard login sequence for Multics, with a few personal features illustrated—abbreviating 'login' to '1' and automatically having mail, or the 'no mail' message, posted to the terminal. Multics is capable of presenting simultaneously an enormous range of different interfaces to its world of users and the following discussion illustrates some of the tools available to build individual environments tailored to either the individual and/or the application.

(Received August 1973)

Multics—Multiplexed Information and Computing Service—is a computer utility jointly developed at the Massachusetts Institute of Technology by Project MAC of MIT, Bell Telephone Laboratories (from 1965 to 1969) and, originally, General Electric, later Honeywell Information Systems. It was implemented initially on the GE 645 and is now available on the Honeywell 6180 computer system, an enhanced relative of the 6080. It embodies many capabilities which are substantially in advance of those provided by many other systems, particularly in the areas of security, continuous operation, virtual memory, shareability of programs and data, reliability and control.

It also allows different programming and human interfaces to co-exist within a single system. This paper describes some of the tools and techniques which are available to enable a user to build his own personal interface or to have one imposed on him by a project administrator.

The requirements for interfaces of a considerably different nature arise for a number of reasons. There is, in some ways, a basic distinction between the needs of a clerical user—perhaps only updating a file or making enquiries in a transaction processing environment—and those of a programmer; and the ways they use the system are quite different. In an organisation where the two distinct classes of user need to share common files and/or programs the basic need for different interfaces begins to emerge. It has been found from experience that the success and ease of use of an on-line system are extremely sensitive to the interface presented to its users.

Also, within the two broad classes of users—clerical and programming—there are subdivisions which are related to the needs, experience and ability of those users. The subdivisions which are found in the first class include:

1. on-line data preparation
2. data acquisition
3. low level access, e.g. enquiry only
4. higher level access, e.g. updating, access to confidential or restricted information
5. computer aided instruction
6. graphics
7. custom built terminals, e.g. ticket printers for seat reservation applications
8. non-programming users of a time-sharing system.

Similarly, in programming environments there are clear cut distinctions between the needs and capabilities of sets of programmers:

1. elementary users and beginners, e.g. school children, first year undergraduates
2. applications programmers
3. systems programmers
4. an APL only subsystem for a university class, etc.
5. programmers developing programs for other systems—for reasons of co-existence or conversion.

This latter point is a major subject in its own right. It arises, for example, if a bureau were to change from another system to Multics; if Multics is used as a software development system for a completely different machine; or if Multics is used to 'front-end' another powerful processor.

In the first of these cases the bureau may not wish to force its users to change directly from the old system to the new, but rather to provide a period when users can continue to use Multics as if it were the older system, and optionally to use the wider 'native' Multics facilities. It is possible to implement a subsystem that precisely imitates the command environment of the other system. The Dartmouth Time Sharing System (DTSS) is implemented in such a way, for example,

Multics as a 'front end'

Here, a physical link can be made between Multics and another system, which might be intended for a specialised 'number crunching' application. Multics acts then as the user interface, the scheduler, the guardian of secure information and a problem solver in its own right. The system can be organised so that users of the complex need have no concern for the JCL of the background machine—indeed they may not even know on which computer the job is run. Alternatively, when submitting a job for a particular machine they could allow Multics to add the control language and pass the job on. Similarly, output can

```
Multics 18-11; MIT, Cambridge, Mass.
Load = 16·3 out of 50·0 units; users = 19

! JJarvis
Password:
You are protected from preemption.
JJarvis PMED logged in: 04#04#74 0616·2 est Wed from TN300 terminal '134'
Lost login 04#04#73 0403·3 est Wed from TTY37 terminal 'none'
new#updated help files: 6180_costs, mpm, lines.
No mail now.
```

Fig. 1

be returned to the users from the background machine.
In such an application, one has the choice of making Multics look like the attached machine--by imitating its command language--or of making the background system (and its JCL) completely invisible.

Multics concepts
There are two important Multics concepts which together allow the shaping of interfaces which are user dependent and at the same time dynamic. These concepts are that of the process and that of a tree structured administrative hierarchy which mirrors the Multics file system.

The process
A process can be regarded as a program in execution together with its address space. A process is created when a user logs into the system and is known by the user's name. It is destroyed when the user logs out. The address space is the set of segments which are available to the user. The segments reside in Multics' virtual memory and the contents are therefore directly addressable, unlike most systems which use virtual storage techniques, where records from data segments must be read into the virtual memory before processing, just as in conventional systems.
During the construction of a process, there are a number of options and restrictions which can be specified by a project administrator which influence and shape the operating environment for that process.

Administration

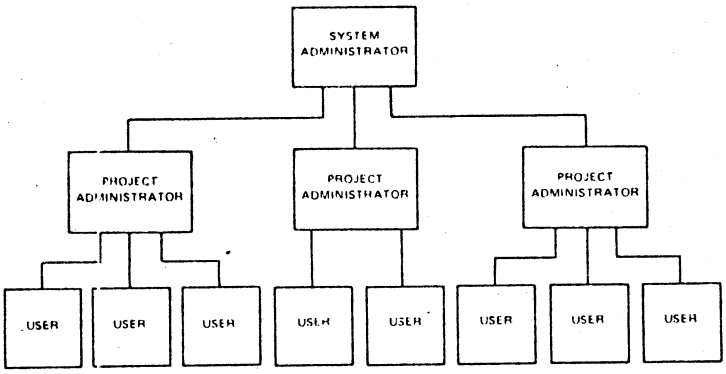

Fig. 2

In the **Fig. 2,** the system administrator (SA) has a wide view of the system as a whole and establishes the operating parameters and prices. He registers projects and project administrators and allocates resource quotas to them.
The project administrator (PA) similarly registers users of this project and allocates resources to them but additionally has considerable control over the manner in which a user may use the system, the resources he may expend, commands available and the general interface which is presented to the user. The PA is in the major position for enforcing alternative interfaces on a user, be he clerical or programming, and some of the parameters he may specify are shown below:

(*a*) Named or anonymous users
(*b*) Non-standard process overseer
(*c*) Allow or force special program executions
(*d*) Provide abbreviations
(*e*) Special command processor
(*f*) Modify search rules
(*g*) Protection rings
(*h*) Home directory
(*i*) Output device interface module (dim).

his home (or initial log-in) directory or changing his process overseer,* which the project administrator may allow the user to choose for himself.
Also, as Multics is, in the main, written in PL/I, the PA is assisted in providing alternative overseers, command processors, etc by the clean and easy interface to which he has access. He may extract existing routines from the standard overseer and modify or include them in the new one just as if he were writing a normal application program.

Making contact and logging in
When a user makes contact with the system and logs in there are a number of alternatives available. When Multics is configured at start up, the physical lines on the system are described and a lines table constructed showing which of the physical lines are to be attached to the initialiser, the system routine which controls the usual logging in procedure. Other processes can attach the other lines to themselves directly. Terminals connected via these lines will therefore by-pass the initialiser and immediately be under the control of a particular program. This could be many things—a transaction processing program or a specialised programming interface.
For a 'normal' line there are then two alternative ways to log in.

(*a*) login — the normal method for a registered user of the system
(*b*) enter — for anonymous members of a project who need not be known to Multics, e.g. members of a short course or undergraduates with minimal requirements.

Process overseer
The project administrator can specify for each of his users which process overseer is to be invoked when that user logs in. The process overseer directly controls the user's interface by defining the commands available and the procedure to be invoked by a given command. The overseer can be:

Process_overseer_	:the default and standard procedure
dart_login_responder	:simulates the Dartmouth Time Sharing System
limited_command_system_	:makes a limited subset of Multics (LSS) available
any_other_overseer	

LSS—Limited Service System—comprises a small set of Multics commands, noted for their light use of resources. The user cannot use other commands and also may have his rate of CPU usage regulated by a governing parameter. A project can therefore ensure that its users cannot exceed a given maximum expenditure during a period but can also know that reasonable access can be continuously available. A professor could leave a terminal logged in and available all day to any of his students without the risk of running out of funds early in a term.
The LSS and variants on it can be used to match the facilities provided to a (programming) user with the capabilities of that user by, for example, limiting the commands available to him or their effect, and progressively widening their scope and power as the user's expertise and experience increases.
Table 1 shows the typical commands currently allowed in the LSS. The command make_commands is provided to enable the project administrator to specify the LSS commands and percentage of CPU time which can be used. The following is an example of an input segment containing the specification of an LSS interface. The first part defines the processor utilisation

*See below.

rate as $\frac{i}{r}$ seconds in a period of i seconds where i is an interval of time and r defines the ratio. In the example the user will be allowed two seconds of processor time in any two minute period.

Example of input file

```
/* set the ratio and interval*/
ratio:          60;
interval:       120;
/*define commands*/
(list ls):   >udd>m>abc>special$list;
logout:         ;
edit:           bsys;
start:          ;
hold:           ;
(pr print):     ;
```

Each line in the second part has two components separated by a colon. The first shows the command (and possible variations) which the user types; the second shows the relative (e.g. bsys) or absolute (e.g. >udd>m>abc special$list) name of the program which corresponds to that command. A blank entry implies the standard Multics command of the same name.

Table 1

Typical commands included within the Limited Service System (LSS) are:

basic
decam—(desk calculator)
delete
edm —(easy to learn editor)
help
list
logout
print
rename

Start-up exec-com

The exec_com, or ec, command allows the execution of a series of commands specified in a segment. The exec_com provides therefore a macro-like facility and includes control lines for conditional execution of commands.

A particular form of ec is the start_up. ec. If a segment start_up. ec exists in the user's initial working directory, the commands and programs specified therein will be executed before coming to command level. This feature is normally used to initialise processes in a particular manner and to cause particular commands to be executed—it is common for users to have their mail or the list of other logged in users printed out in this manner, as in the login example in Fig. 1.

The start_up. ec also allows the project administrator to force a user into a particular environment. The PA can:

(*a*) specify a start_up. ec which will control the interface
(*b*) prevent a user from by-passing the start_up. ec (an attribute normal users have)
(*c*) further restrict the user, e.g. within the ec he can disable the 'quit' or interrupt button to stop the user 'forcing' his way out of the environment.

The interface presented to the terminal user in this manner could, for example, be the use of a proprietary software package where input is invited from the terminal, or a conventional transaction processing system. If the latter case were, for example, an order entry system, the initialised program could begin with an invitation to:

ENTER NEXT CUSTOMER NUMBER:) (
ENTER E IF END OF BATCH:) (

and could continue from there without ever going to Multics command level.

Command processing

Users who proceed to the command processing level can still choose—or have forced on them—one of many further interface-shaping options.

The listener procedure reads commands from terminals, edits them in accordance with conventional keyboard erase and kill characters, and converts them to a canonical form. The listener then calls either the command processor directly or the abbrev command processor.

Abbreviations

Any user can set up his own abbreviations at will and change them at any time. The command abbrev, which can be abbreviated to ab, must be invoked before personal abbreviations are accepted. Most users therefore include the abbrev, or ab, command in their start_up. ec (see above). This use is in addition to the system command abbreviations, e.g. the command

'how many users' has a standard abbreviation of 'hmu'.

A user could, if he wished, abbreviate it still further, perhaps to merely h.

Abbreviations can be applied to system commands (as above), a user's own program names, directory names, his own name (e.g. jj for JJarvis)—in fact up to eight characters for any string of up to 132 characters.

This ability allows a project administrator to provide project wide abbreviations, perhaps to facilitate imitating other systems—'bye' for 'logout' for example—or even for other languages—'adios' for Spanish speakers, etc. There are very interesting areas of application here for multinational companies wishing to ease the use of a central system by the component countries.

Command processor

The standard command processor can be called by the listener, by abbrev after expanding an abbreviation, or by any other command which examines terminal input for recognisable commands. Command processors are called through the 'cp' entry to the command utility subroutine. A complementary entry, set_cp, allows the project administrator to set an alternative command processor. Subsequent calls to cp will be passed to the new command processor. This technique can therefore be used by a subsystem developer to provide a purpose built transaction processing interface for a clerical user or to ensure that the new subsystem retains control while allowing the use of many standard system commands.

Search rules

When the command processor receives a pointer to the name of a program to be executed, it follows a set of search rules to locate the named segment. (In passing, Multics binds external references to both procedure and data segments at execution time. The 'dynamic linker' which effects this also follows the same search rules.) **Table 2** shows the default search rules.

Table 2 Search rules

1. Initiated_segments	check the already initiated segments;
2. referencing_dir	search the parent directory of the segment making the reference;
3. working_dir	search the working directory;
4. home_dir	search the home directory;
5. process_dir	search the process directory;
6. system_libraries	search the default system libraries.

Both users themselves and project administrators can specify their own set of search rules. The commands

set_search_dirs (ssd)
and set_search_rules (ssr)

are available for altering the rules. Alterations may be made by changing the order shown in the default list, by adding or inserting further directories in the list, or by omitting certain items. A project administrator can leave system libraries out of the list and thus make many standard commands unavailable to a user; he can similarly insert a special project directory high in the list so that the user will be presented with alternative (perhaps less expensive) versions of a standard command. As an example of this, he may provide a Basic compiler which uses a subset of the main language specification for the use of school children or beginners.

An easy way for the administrator to change the rules is to provide a start_up. ec (see above) for his users and to include the command

ssr project_search_rules

in it, where project_search_rules is the name of a segment containing the new rules.

Terminal input and output

The input/output system in Multics is oriented around the concept of streams. Usually the output to a terminal is directed to the stream user_output. A stream is associated with or attached to a particular device interface module (DIM) and corresponding device, and the attachment can be changed dynamically if required.

The use of intermediate interface modules can be interposed in this chain to carry out additional processing for a particular user or terminal type.

A common use of these facilities allows the easy attachment of new terminal devices with, perhaps, slightly different characteristics or applications requiring heavily formatted data. The approach also allows for the translation, on both input and output, from and to different languages. For example, an international company, with terminals throughout Europe, connected to a central system, could provide a local interface where common files and procedures could be accessed in English in England, French in France and so on, and have the system look after the translation as appropriate. This is an extreme, and perhaps unlikely, application but entirely possible and the approach is of value in certain circumstances.

New commands

As an illustration of several of the aspects described above, let us consider the definition of a new command to compile and execute a FORTRAN program. On many time-sharing systems the command RUN, with various options, is used for this purpose, and we can trivially develop a RUN command for personal or project use as follows.

Assume a source program segment has been created and named 'segname. fortran.' To compile this a Multics user types

fortran segname (or ft segname)

and to execute it after a successful compilation he types

segname

(The segment 'segname' is created by the system if the compilation is successful.)

By using the exec_com command, a series of commands listed in a segment with suffix .ec may be executed. If the segment ftrun. ec contains

fortran segname
segname

then the command

ec ftrun

will cause segname to be compiled and executed. This is of limited value as it stands, but by the use of parameters in ftrun. ec we can generalise it, e.g. let ftrun. ec contain

fortran & 1
& 1

then the command

ec ftrun segname

will cause segname to replace the occurrences of & 1 in the segment and it will be compiled and executed as before.

We now use abbrev to abbreviate ec ftrun as RUN thus:

.a RUN ec ftrun

(.a is the abbrev control request to define a new abbreviation), and thereafter the user can compile and execute any FORTRAN program by typing

RUN source_segment_name

This, then, is a simplified example of one of the ways in which other systems and commands can be provided for reasons of compatibility, co-existence or familiarity. In practice, of course, several other parameters would normally be involved, and other commands would not necessarily be as straight-forward.

Rings of protection

The Multics ring protection, implemented in the hardware, has been described at length elsewhere (Graham, 1968). It is a refinement of the general access controls (permission to read, write and execute segments—also hardware implemented in Multics) and allows subsystem writers to develop much more secure applications than would otherwise be possible.

It can be pictured as a series of concentric circles, as **Fig. 3.** Ring 0 has the highest privilege and ring 7 the least. The normal Multics user operates in ring 4, but the actual rings in which a user may work are, like so many things, specified by the project administrator. The user will probably be unaware that he is constrained to a given ring or set of rings, unless he attempts deliberately to cross a ring boundary. Segments reside entirely within a ring and access from a ring to a data segment, for example, in one of higher privilege is possible through 'gates', or entry points to a procedure in the higher privilege ring. The procedure can then sift the data as appropriate and provide all details, a summary only or nothing at all depending on the user and/or his ring.

Salary and medical history databases are often quoted as examples where the ring structure is valuable. A list of employees, salaries, work history, etc can be maintained in, say, ring 4. Some employees might need to have access to the list of names and addresses only, others might be allowed to know the total monthly salary bill but not the specific salary of each employee. The rings allow the easy construction of this and similar environments.

Again, a junior clerk, perhaps in ring 7, can use the same enquiry code as his supervisor, in ring 6, but be given only a subset of the information which the superior would receive. Similar applications are currently implemented using elaborate systems with locks and keys which the supervisor carries. The structure gives the same sort of control more flexibly without requiring special hardware and terminals with locks.

The user

The programming user can either have all Multics native facilities available to him or, as we have seen, have them augmented, restricted or modified to look like another system—in this regard he is under the project administrator's control.

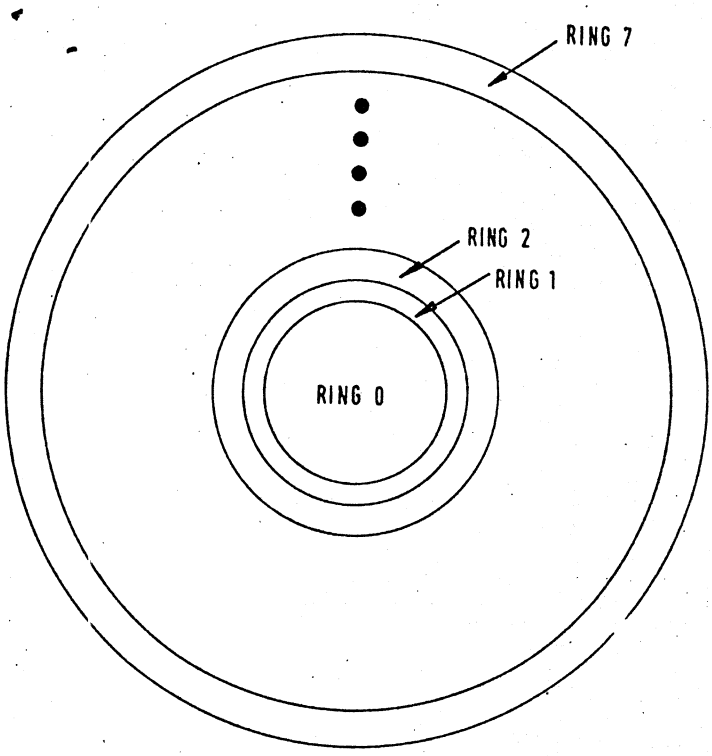

Fig. 3

The user can use many of the facilities himself to further alter his interface for, usually, reasons of convenience and familiarity. He may, among other things,

(*a*) vary his process overseer
(*b*) specify his normal working directory
(*c*) use or escape from his start_up ec
(*d*) receive brief or normal messages
(*e*) elect not to receive ready messages at the completion of each command or ask for monetary totals rather than cp time to be shown
(*f*) use abbreviations
(*g*) vary his search rules
(*h*) use alternative DIM's
(*i*) move his base of operations to different directories of his own or other users (with proper permission).

Available systems

Currently implemented and available as alternative operating systems under Multics are:

DTSS —The Dartmouth Time-Sharing System implemented by Dartmouth College, Hanover, New Hampshire.

GCOS—The standard operating system for the Honeywell Series 6000 and Series 60/Level 66 ranges of computers.

These have major differences from native Multics. The Limited Service System (LSS) which is available is a proper subset of Multics. Other operating system interfaces, for completely different systems, are under development.

PL/I

The very great majority of the Multics operating system commands and subroutines are written in PL/I. Apart from obvious benefits in speed of implementation, documentation and maintainability, it eases the task of subsystem writers who are given the ability to interface directly and easily with the operating system itself.

Systems programmers can and do build their own experimental Multics systems, which run during normal service, to test new modules without requiring the complete system, or running the risk of ruining parallel production work.

Conclusion

The above brief descriptions of many of the environment shaping tools available indicate the wide choice available to the project administrator and subsystem designer who wish to provide a new Multics face for the outside world.

Major significant aspects of the approach are:

(*a*) there are no supervisor modifications required to implement the possible multiplicity of external faces
(*b*) no operations involvement is necessary: alternative interfaces are remotely implemented, maintained and administered
(*c*) they can be dependent on the project or the user himself within a project.

```
logout
JJarvis PMED logged out 04#04#73 0653·9 est Wed
CPU usage 17 sec
hangup.
```

References

GRAHAM, R. M. (1968). Protection in an information processing utility. *CACM*, Vol. 11, No. 5, pp. 365-369.

SCHROEDER, M. D., and SALTZER, J. H. (1972). A hardware architecture for implementing protection rings. *CACM*, Vol. 15, No. 3, pp. 157-170.