User control in a multi-access system

By A. G. Fraser*

Administrative controls in an operating system must involve decisions about user status and permitted activity. The mechanisms are traditionally *ad hoc* and reflect the multitude of curious relationships which usually exist in any organised society. It is also usual to find them distributed through the software complex. This paper suggests a centralised and uniform approach in which a common notation is used to describe all conditions governing status and privileged activity.

(First received October 1967)

The multiple-access system for the Titan computer at Cambridge contains a variety of controls which discriminate against some users to the presumed advantage of others. Controls of this type are an essential feature of any operating system that supervises a shared facility, although it is usual to find that the detailed design is particularly *ad hoc*. The Titan system is not unusual in this respect and one can observe a variety of different control mechanisms each responsible for exercising control over a different sphere of activity. In this paper I suggest that the control mechanism should be centralised and that the rules of discrimination should be concentrated at one point and held in a form that facilitates change.

Imagine a multi-access system in which there is a disc-based filing system providing time-sharing facilities simultaneously to a number of different users. In addition to work-load administration, all I/O operations are handled centrally. Some of the operating system runs in a special privileged mode but as much as possible works in the mode normally associated with user programs.

The system will give controlled access to files so that one user's property is protected against unwarranted interference or inspection. It will also require evidence of authority when one user attempts to create a file that is to be charged to someone else. A password, or similar device, will be used to prevent one person masquerading as another and special checks will need to be made to prevent one person discovering another's password.

Those parts of the system which operate as normal user programs will need to use facilities that are not available to the general public. There may be an incremental file dumping program which will need to be able to read all files on the disc, and the I/O routines will require similar facilities. The work-load scheduling program (hopefully a centralised function) will also need special privileges if it is to be able to monitor system performance.

There will undoubtedly be flaws in the security provided, and arrangements for some form of policing will be needed. Similar requirements will be made by the management who will wish to monitor user activity as well as instigate changes. Finally, a special 'way in' to the system will be required to deal with corrupted passwords or malformed system files.

In each of the above examples of control, the system must decide whether or not to allow a user program to proceed with a requested activity, and this decision will be based upon some special characteristic of the job, the circumstances that surround it, or the information on which it is operating.

Central control

In a number of the examples quoted above, the control could be described as a restraint on some user, but this is not exclusively so. The incremental dump program requires privileges but cannot be identified with a particular human being and there are utility routines that require privileges but which themselves are used by the non-privileged user. So it was with the Titan system, which associated privileges with users, that we ended up by inventing 'pseudo-users' and adopting other unsatisfactory expedients.

Expressed in its simplest form, a control system should be based upon a free standing list of declarations of the form:

'Allow (or deny) activity A if Condition C is True'
. . . (1)

I shall refer to this as the *Activities List*.

A central control routine should be used to process this list and answer the question:

'Is program P allowed to perform activity A?'
. . . (2)

Furthermore the mechanism should be designed to be free from unnecessary restraint so that further changes to the system may be contemplated.

It is assumed that each activity can be identified by a unique name. In many cases it will be convenient to use a mnemonic since activities are not usually identified by simple names. Some mnemonics could be:

DISCABS	Reference to the disc store directly using actual hardware addresses.
OWNER CR	Creating a file directory for a new file owner.

* *University Mathematical Laboratory, Corn Exchange Street, Cambridge.*

READNOCHK Access to a file without satisfying the usual privacy checks.

In order to build an activities list the operands used in the conditional expressions also need to be identified. Two of the operands used in the Cambridge system are:

USER The unique identity of the individual responsible for a single computer run.

COMMAND The unique identity of the program being obeyed (for example: FORTRAN COMPILER).

The following is a possible section of an activities list:

1. Allow DISCABS if COMMAND = 'FILE MANAGER'.
2. Allow DISCABS if (USER = 'BILL' OR USER = 'HARRY') AND COMMAND = 'DISC POSTMORTEM'.
3. Allow READNOCHK if COMMAND = 'INCREMENTAL DUMPER' OR ((USER = 'BILL' OR USER = 'HARRY') AND COMMAND = 'FILE POSTMORTEM').

This list will be held in the computer while the system is running and will be scanned by the control routine. It is therefore essential to use a stored format that facilitates rapid processing by the control routine. In the Cambridge system the activity list contains fixed length entries and each conditional expression fits into a pre-determined structure. The list is held in one of the files that is used by the operating system itself, but there are special arrangements that allow the administration to update it while the system is running. As is usual in these circumstances, we have chosen to compromise between full flexibility and rapid processing.

Program status

Now, we may also observe that the Activities List contains activity names and conditional expressions and that, so far, nothing has been said about the forms which individual terms of the expression may take. For example, a single term might be a Boolean function and unnecessary duplication could be avoided by using appropriate function definitions. Thus a Boolean function called SYSTEM PROGRAMMER might be defined by the expression:

USER=BILL or USER=HARRY

The Activities List would then contain entries which include references to SYSTEM PROGRAMMER, for example:

Allow DISCABS if SYSTEM PROGRAMMER and COMMAND = 'DISC POSTMORTEM'.

Functions of the type described above will, henceforth, be called *Status Declarations* and a central list of these will be called the *Status List*. Its entries will take the form:

'Allow (or deny) status S if condition C is true.'
. . . (3)

The procedure for handling this list will be almost the same as that required for the Activities List, and so a second entry to the control routine will be used to answer questions of the form:

'Is user P entitled to status S?' . . . . (4)

I shall assume throughout this paper that the status term S carries no parameters, although such an extension would be consistent with the scheme described here.

Composite activities and status

System performance can be further enhanced by the use of expressions in which distinct activities are identified. For example, the expression ($A1$ and $A2$ and $A3$) would represent a composite activity which involved the three distinct activities $A1$, $A2$, and $A3$, and it could be used in place of the single activity name in one Activity List entry. In this case the single entry would authorise any or all of the individual activities involved.

This device can also be used to advantage in the Status Lists so that declarations of the following form would be permitted:

'Allow status ($S1$ and $S2$ and $S3$) if .....'

By using activity and status expressions in this way one can hope to save space in the respective lists, but system performance can also be improved by making composite activity and status demands acceptable input conditions to the control routine. Entries of the following type could be permitted:

'Is user P entitled to status ($S1$ and $S2$)?'

In the system used at Cambridge the activity expressions are represented by Boolean vectors in which there are as many elements as there are possible activities. Condition evaluation is thereby conveniently interpreted in terms of logical operations on these vectors.

Remembered decisions

It is normal for an operating system to seek increased performance by remembering certain information about a user program even though that information could be re-computed at will. Thus, for example, it may choose to determine the truth of the term SYSTEM PROGRAMMER at the start of a run and thereby avoid subsequent re-evaluations.

An almost analogous situation exists with information that is more difficult to re-constitute. When a user first logs into the system he may be asked to prove his identity by quoting a password, or something similar, and the system would remember that this had been done satisfactorily. The user would be given status IDENTITY PROVED. If subsequently the user asked to use a restricted activity the system would associate the value TRUE with any reference to the status IDENTITY PROVED.

If the validity of a status decision can change there are dangers involved in the use of remembered decisions.

As with all redundant information of this type, care is required if the source of the information itself changes.

One solution would be to make the central control routine distinguish between three values of a remembered status decision: True, False and Uncertain. Then, if the status were uncertain the control routine would evaluate the status before proceeding, and it would record the decision reached in order to avoid subsequent evaluations. With this mechanism in force, one could simply arrange to mark appropriate remembered status decisions as uncertain when a change in circumstance threatens their validity.

In the Cambridge system many of the remembered status decisions take only two values: True and Uncertain. This means that successful control checks can be made quickly but more work has to be done when a user attempts to do something illegally. Program speed is also enhanced by establishing certain status names that directly correspond to activity names. For example, the status name S-READNOCHK might be used in the following activity list entry:

Allow READNOCHK if S-READNOCHK.

Once a user has established the status S-READNOCHK, permission to access a file can then be granted with minimum delay.

There are two further activities which are used by commands that involve remembered status decisions. One checks the status value, causing it to be remembered, and the other unsets a remembered status. No separate attempt is made to unset a status value during the execution of one command even when the conditions which led to that status setting themselves change.

Control decisions are not always made with a frequency that is high enough to justify special treatment, and the price paid for a central and general purpose mechanism may therefore be quite modest. For the few control decisions that are made with high frequency, the device adopted at Cambridge is based upon that described above; a single status is associated with the activity concerned so that a single test is all that need be made on most occasions. In these special cases the single test can be made outside the general purpose routine and the latter will only be called into play if the test fails.

Facilities and property

It is invariably the case that a general purpose procedure is less efficient than a specialised routine, but the actual costs may often be small in comparison with the total operating cost or the advantage of a compact and flexible design. However, there is one form of specialisation that may be worth making. It involves a distinction between restrictions on the use of communal facilities and restrictions that ensure the privacy of private property. This distinction is, of course, only a matter of convenience since control is always exercised when a program requests a particular activity.

Files, magnetic tapes, and exchangeable discs are examples of items that may be treated as private property. The system will maintain a directory of such items so that they may be identified and serviced automatically. In addition each will have a possibly unique set of conditions that govern its use, although the number of distinct modes of use will be limited. In general, there will be a complete set of conditional statements for each object and these will define the acceptable conditions under which each activity type is allowed.

One might also expect a significant traffic in new and discarded objects so that the system directories will require regular updating. Similarly, the set of conditional statements associated with each object will need to be deleted or extended. But since both operations are usually linked to one event (e.g. file access) there would seem to be some value in combining the Activity Conditions with the conventional directory entry. Fortunately this can be done without loss of facility since the particular activities involved will only be relevant when one particular directory is being accessed. But some facility will be lost if the directory entries are not regarded as an extension of the central Activity and Status Lists or if a separate routine is used to process them.

The system design could therefore be as follows:

'Each property directory will contain statements of types (1) and (3) where these statements apply only to the particular set of property administered by the directory concerned' . . . (5)

The control routine will be furnished with two more entry points which will combine the central lists with the lists contained in one property directory in order to answer questions of the form:

'Is user P allowed to perform activity A on property governed by entry E in directory D' . . . (6)

and

'Is user P entitled to status S when handling property governed by directory D' . . . . . (7)

New control statements

Whenever a file is created a new set of conditions will be added and these will control the various ways in which the file can be used. The ability to create a new file would necessarily imply the ability to specify these conditions. To change the conditions after the file has been created will be regarded as one mode of file use and should be controlled accordingly.

From time to time the central list of Activities will need to be changed and this will need close control. But there is no reason why this control, like all others, should not be exercised by the central mechanism itself. The act of updating either the central Status List or the list in one property directory would be a recognised activity and would be subject to the appropriate conditions.

The right to create new entries in the Activities List will itself be subject to a control that dictates the type

of conditions that may be used as well as the circumstances under which extensions are permitted. Similarly each entry in these lists will be subject to different forms of restraint; the leader of a group can change its membership but cannot add a new group. For this reason, the status entries and the entries in the central activities list should each have one associated condition that determines when updating is allowed. In this respect property descriptions, Activity descriptions and Status descriptions are treated similarly.

Recursion

I have already suggested that a status description plays the role of a Boolean function and can be compounded into any conditional expression, and I do not consider it necessary to prohibit one function from using another. However, the structure of the control lists must be restrained in order to guarantee that the system does not come to a loop stop. For this reason recursion must be ruled out.

It is probable that, during the evaluation of a conditional expression, the system will be asked to perform a restricted activity. Therefore, to avoid any chance of a non-terminating sequence of checks, certain restriction must be lifted while evaluating one condition. At Cambridge this result is obtained by setting the necessary status values when the control routine is called. In fact, the operating system holds a short list of routine names and with each are details of status values that are unconditionally assigned when the routine is called.

Now, the lists of activity and status checks will be extendible, and it is necessary to devise a mechanism for doing this that does not interfere with system performance. In particular, it must not be possible for someone to devise a 'pseudo' status entry that takes advantage of the restriction-free mode in which conditions are evaluated. To a substantial extent this risk can be diminished by removing any possibility of side-effects so that the only product of evaluating a status condition can be a single Boolean value. A further restraint would be provided by restricting the notation so that only certain data can be used; but care is required if one is to avoid loss of useful flexibility.

The Cambridge system

That part of the Titan Operating System that could conveniently be modified has been re-written to use a central control routine. Each of the activities controlled by this routine is given a unique number, n, and the internal identifier of that activity is the single computer word in which the nth binary digit is non-zero. For example, the value 00100000 identifies activity number 3 which might be the act of adding a new file directory to the system.

There is one central directory in which a typical entry E_i contains a predicate P_i and a binary value A_i. If P_i is true for some user, then the value of A_i is the logical sum of the identifiers of activities permitted to that user. For example, if P_i is equivalent to "User name = 'FRED' " and if $A_i = 0010010000$ then the user called FRED will be allowed to perform activities numbered 3 and 6. Of course, the predicate P_i is held in a compact form that permits rapid evaluation and the range of predicates is strictly limited. Each contains at most four terms which can only be combined using the logical operator AND. Each term must be chosen from the following list (X is a parameter specified by the user):

USER NAME = X
USER OBEYING COMMAND X
USING FILES IN GROUP X
HAS QUOTED KEY X
USER HAS QUOTED HIS OWN PASSWORD
USER IS SITTING AT A CONSOLE

To decide whether a user can perform an activity with identifier r, the system scans the central directory and computes the logical sum of all A_i for which P_i is found to be true. This value is then compared with r and the activity is authorised if the logical product of these two values is not entirely zero. Expressed in another way, the action is permitted if

$$r \Sigma P_i A_i \neq 0 \quad (8)$$

In addition to the central directory, there are a number of file directories in which a typical entry can either describe a file or an authority. The entries which describe authorities are precisely the same as those held in the central directory. The file entries contain information that is required for the proper administration of the file system together with a value, s, which dictates the extent to which the file is available for use. The value of s is the logical sum of the permitted activities and its value is usually set by the user when he creates a new file.

Permission to perform activity r on an existing file is only given to someone who is entitled to perform that activity by virtue of authorities vested in him and provided that the activity is also included in s. The action is therefore permitted if

$$rs \Sigma P_i A_i \neq 0 \quad (9)$$

The summation in this expression is performed over entries in the appropriate file directory so that the file owner can build up his own list of special relationships with other users. Expression (8) is also evaluated over the entries of a file directory when the activity involves that directory. In all other circumstances the expression is evaluated by scanning the central directory only.

The five ways of accessing an existing file (Update, Delete, Change S, Read, Execute) have each been given four distinct names in order to obtain greater variety in the distribution of authorities. The identifier of Read File is 00010 00010 00010 00010 and that of Execute File is 00001 00001 00001 00001. Also, by convention, the authority 11111 00000 00000 00000 is given to the directory owner and the authority 00000 00000 00000 11111 is given to everyone. With these arrangements, the file owner can choose s so as to make individual files available in different ways to different classes of user.

Conclusion

The change to a central control (in so far as it was attempted) seems to have been very successful and removed many of the curious *ad hoc* restraints that previously caused trouble. The ability to re-define the conditions which determine the use of central facilities and the greater freedom which stems from a simpler and more uniform approach have proved more useful.

Although the use of explicit conditional expressions provides greater flexibility and makes for simpler software, it seems to be necessary to provide some special purpose commands that use the basic facility to provide specialised services. Not every user wants to use the system in its most general form; many would value a special purpose tool. Indeed, it may be that the value of centralised control as described here is just that it allows easy tailoring to special situations, a design objective that would benefit most manufacturers of software.

Acknowledgement

The invaluable opportunity to make mistakes and learn by them was provided, in this instance, by the Titan Multiple-Access project at Cambridge University under the direction of Professor M. V. Wilkes. The work was supported by the Science Research Council.

Obituary

Dudley W. Hooper, M.A., F.C.A.

Dudley Hooper, whose death occurred on 12 January 1968, was born 57 years ago. He was educated at Charterhouse and at Clare College, Cambridge, and qualified as a Chartered Accountant in 1935. For 32 years he then specialised in management and organisation projects, with special reference to mechanisation and, later, E.D.P. During the years 1940–45 he was on war service, mostly in Africa, and much of this time he spent as a staff officer on commisariat organisation work. He then served as secretary or accountant to a number of organisations and, after 6 years with the National Coal Board, became the chief organising accountant to that body, and was responsible for the development of the Board's computer projects. In 1964 he was appointed Technical Officer to the Institute of Chartered Accountants in England and Wales.

But meanwhile he had lectured over a wide field, notably at the Northampton Polytechnic (now the City University) together with Mr. A. Geary and Mr. M. Bridger, and in 1956 he led the little party of zealots in forming the London Computer Group. A year later the British Computer Society came into being, and Dudley Hooper was its first Chairman. But behind this simple statement was in fact an extremely difficult period which called for all those personal qualities with which Dudley Hooper was fortunately endowed. These included the wisdom of Solomon and the patience of Job. Difficult as it was to get the Society off the ground, no-one could then have foreseen that within 11 years the Society would have grown to its present membership of some 18,000. Dudley's dedication was phenomenal; travelling constantly as he did he still made time for conferences, lectures and editorial work, and the first Memorandum and Articles of Association of the Society were written by him personally. He became President of the Society in 1961–62 and continued to take an active part in the Society's affairs. He was indeed a member of the Editorial Board of this *Journal* up to the date of his death, and was the founder and first Editor of the *Bulletin* with valuable assistance from his wife.

Dudley Hooper was widely known for his papers and his lectures, not only to technicians but to students. His personality was genial, modest and engaging—he always

spoke at the level of his audience, without pedantry, and he wore with pride a tie presented to him by the Chartered Accountants Students Society. Indeed Dudley's last published work was a booklet for the General Educational Trust of the Institute of Chartered Accountants in England and Wales entitled *The Computer as an aid to Management*. It went to press shortly before his death, and has thus been published posthumously. The publication announcement was followed immediately by an overwhelming demand for copies. This is an indication of the width of the field in which Dudley Hooper will be so sorely missed, and of the debt owed to him by the fields of management (a subject which he taught by precept and example), the accounting profession, of which he was so distinguished a member, and the world of E.D.P., in which he was so noteworthy a pioneer. He occupies an unique place in the history of E.D.P., and in the minds and memories of those whose pleasure and privilege it was to work with him.

E. E. Boyles