Application Development Aids

Burroughs

Technical Information Paper

DATE: 15 October 1979

Origin: International Group

PRODUCT: DMSII

SUBJECT: DMSII EXCEPTION HANDLING

The attached paper on DMSII exception handling by B. Johnson, Melbourne, Australia is reprinted here courtesy of L. Aubé, International Group.

TO: BMG Communications Code **RC**

Printed in U.S.A.

1114238-013

DMS II EXCEPTION HANDLING (B. JOHNSON, JANUARY 79)

1. INTRODUCTION

In most programs, to do the majority of the processing is fairly routine and easily accomplished. Much more work, however, is required to handle the multitude of exceptions that may occur.

For example, reading input data is a fairly trivial task, but the validation of that data against the design rules may take many pages of code.

This document discusses how, in the same way, a DMS II program must pay careful attention to the processing of the exceptions that can occur when using DMS II host language verbs.

2. EXPECTED AND UNEXPECTED EXCEPTIONS

For each DMS II host verb, a variety of circumstances may interfere with its success.

In some circumstances, such a condition is expected, probably the most common example being a sequential scan (FIND NEXT) of a data set or set until a NOTFOUND exception indicates there are no further records in the data set.

However, unexpected exceptions are an indication of abnormality in the system, and appropriate action should be taken.

To ignore such conditions could be likened to accepting input data that fails validation criteria. Indeed, proper exception processing is more important, from a system viewpoint, than the use of the DMS II verbs themselves. To ignore such conditions when the system is attempting to make them known could place the integrity of the data-base at risk, at both the physical file and logical relationship levels.

.... /2

DMS II EXCEPTION HANDLING

2. EXPECTED AND UNEXPECTED EXCEPTIONS (cont'd)

To illustrate further, consider the following two classes of unexpected exceptions:

2.1 IOERROR Exceptions

IOERRORs generally represent a storage failure, and signal a need for some form of data recovery.

Such an exception occuring in the following -

```
LOCK NEXT ... ON EXCEPTION GO TO NO-MORE-RECORDS
```

will generally imply that a record that does exist will not be used. This could easily lead to logical inconsistencies in the data held within the database.

2.2 SYSTEMERROR Exceptions

The occurrence of this exception usually indicates that the DMS routines have detected some form of internal inconsistency in their processing, implying system software or hardware errors that may threaten database content integrity.

This class of exception should not happen very often, and indeed is not expected to occur. But, if it does, appropriate action needs to be taken quickly to minimise any impact it may have.

3. "IMPOSSIBLE" EXCEPTIONS

A further reason for omitting processing of particular exceptions is because it is thought the condition could not occur in the given circumstances. However, it could be argued that this is why it should be included.

One of the best indications of error in either program logic or system software is by the occurrence of "impossible" conditions.

It should be pointed out that with proper coding, the overheads of this extra checking should be fairly minimal. By testing for the most common condition first, the extra code for other conditions will not be executed unless needed; the extra code space required should be a small price to pay for the increased robustness of the program, assuming one wanted to dispute its necessity in the first place.

DMS II EXCEPTION HANDLING

4. DEADLOCK PROCESSING

It appears that one of the most misunderstood exceptions is DEADLOCK. Many programs seem to take the attitude that "it cannot happen to me", which is begging the fate according to Murphy.

The authors of many update programs similarly bypass this exception because there will only ever be one copy of the program running at any point in time. Again, Murphy's preachings are overlooked, the update schedule starts running behind, and... Theoretically, a DEADLOCK may be indicated by any LOCK or BEGIN-TRANSACTION statement, especially if program or system errors are to be catered for.

A DEADLOCK may occur with only one program running if it multiply invokes a particular data set, or it may occur simply because several programs in the mix are LOCKing records.

The system recognises two levels of DEADLOCK detection. The first is obvious; program A with record X LOCKed wants to LOCK record Y, whilst program B with Y LOCKed is attempting to LOCK X. An obvious stalemate condition exists, and the system acts to break the DEADLOCK.

The second level of detection is the recognition of a prolonged state of contention. Simply stated, if a program waits for a LOCK on a record for more than a specified period of time, a DEADLOCK by contention is returned to the user program. This means that someone else has had the record locked for longer than that period without releasing it. This is significant for real-time work.

Currently, this time period makes use of MAXWAIT task attribute on B6000/B7000 Series DMS II.

An important effect of receiving a DEADLOCK exception is that all records currently LOCKed by a program will have been implicitely FREEd. Hence, if a program is to handle this exception, one of the requirements will be to reLOCK all the previously LOCKed records, not just the record receiving the DEADLOCK exception.

A further implication of DEADLOCK detection should be recognised. If a hitherto LOCKed record is FREEd in this way, the program should allow for the possibility that its contents may be changed by the second program before the first can reLOCK it. If decisions have been made on the record contents, these will not be valid. If some other records have been updated as a function of its contents and there are still more to be done, the later records may now need to be updated according to a possibly different set of criteria.

5. EXCEPTION DETERMINATION

On small systems (B1700/B1800), the system only returns to the program the category of the exception encountered.

However, on large systems, three parameters are returned to the program to allow greater refinement of the nature of the exception. These are as follows:

5.1 The category is the major parameter, and is used to specify a major class of error.

5.2 The ERRORTYPE represents a subcategory and in many cases is quite predictable. For example, NOTFOUND category on FIND NEXT on a set should produce an ERRORTYPE of 2 whilst NOTFOUND on FIND PRIOR on a set should be 3. In other cases it may be less obvious, such as in the case of IOERROR. Depending on the hardware failure, any one of a number of subcategories may occur.

Note that a test for NOTFOUND on a FIND NEXT on a set would still do well to ensure the correct subcategory of 2 for integrity reasons, as another value could be indicative of a more serious problem.

5.3 STRUCTURE. The third parameter is the STRUCTURE number on which the error occurred. As this is not always the structure upon which the verb is directly acting (for example, a STORE might return a DUPLICATES on an automatic SET or SUBSET), it may prove useful to examine this item.

5.4 USE OF DMTERMINATE. DMS II provides the three parameters as a definition of the condition which occurred. If an abnormal condition occurs it is desirable that some indication of all three values be provided as a complete definition of the exception. The DMS II Verb DMTERMINATE, available since II.9, provides this, and may be suitably utilised.

6. RESPONSIBILITIES OF THE DATA BASE ADMINISTRATOR

The responsibilities of a Data Base Administrator (D.B.A.) have been much discussed elsewhere, and no attempt will be made here to add to that discussion. Rather, the simple assumption will be made that the D.B.A. is responsible for the integrity of the data and relationships represented in the data base.

This being the case, the effectiveness of exception processing in an application program will be of prime importance to the D.B.A. Poor exception processing will be the biggest threat to the integrity of HIS Database.

6. RESPONSIBILITIES OF THE DATA BASE ADMINISTRATOR (cont'd)

It is for this reason that the D.B.A. should establish some level of standards for the exception processing of programs accessing the database. He should not allow access to any program that does not meet those standards because by so doing he would be compromising the integrity of His Database.

The enforcement of such standards may take a variety of forms. It may simply be a document to which the programmers continually refer whilst coding the program; it may be a series of standard library procedures to be used by the programs, at either compile or run time (that is, either symbolic to be included into the program source, or a library of precompiled procedures to be called by the application program).

On the basis that the D.B.A. is not always a programmer, probably the former approach is the preferable. In either case, the program design will need reviewing at the end of development to ensure standards are met before going into production.

The standards, by definition, must be comprehensive. The exception processing will generally need detailing in conjunction with other aspects of the system, not just as a standard on its own. It will probably vary from application to application.

Exception processing is a vital part of the system; it is not meant to be tacked on as a frill.

7. REFERENCES

The following parts of BURROUGHS B7000/B6000 DMS II HOST REFERENCE MANUAL (# 5001498) are of particular importance to exception processing formulation:

(a) Section 6, details methods of incorporating exception processing into a user program, as well as usage of the three parameters provided by the DMS II System.

(b) P. 14-12...documents the DMTERMINATE verb.

(c) Appendix B...lists the categories and subcategories that may be returned to a program as a result of a DMS II verb. In particular, p.B-8 provides a table cross-reference of the categories applicable to each DMS II verb. This should probably be the most important single page in the manual for the application programmer.

7. REFERENCES (cont'd)

It should also be noted that an "INVALID OPERAND", rather than an exception condition, results from attempts to:

(a) access a database before opening it

(b) modify an audited database outside transaction state.

ATTACHMENT 1 SMALL SYSTEMS DIFFERENCES

The following notes attempt to identify the points in this document that differ with small systems. DMS (relevent to VII.0):

(i) Small systems exceptions produce only a CATEGORY for an exception; ERRORTYPE and STRUCTURE are not provided.

(ii) DMTERMINATE is not available on small systems

(iii) The time period for DEADLOCK by contention on small systems is 3 minutes.

(iv) "INVALID OPERAND" interrupts in large systems DMS (see 7 REFERENCES), the first case produces an OPENERROR and the second an AUDITERROR exception.

(v) LOCK and MODIFY are exact synonyms; either may be used on Large Systems whereas only the latter is provided on Small Systems.