To: MTB Distribution

From: Gary C. Dixon

Date: January 13, 1978

Subject: Generation Data Sets on Multics

THE PROBLEM

Many Multics sites run commercially-oriented job streams which typically apply transactions to one version of a data base to create a newer version. In this way, bad transactions applied to a data base can be removed by deleting the new version.

As the number of data base versions increases, the problem of identifying the contents of a particular version, and of accessing the newest version (or second newest version, etc) becomes cumbersome, especially if programs which reference the files are driven by exec_coms which pre-attach the files prior to invoking the program.

Currently, the only method for referencing different file generations from an exec_com is to edit the file's pathname or tape file parameters which appear in the attach description of io_call commands in the exec_com. This is cumbersome, error-prone, and requires extreme care that the correct version is given before running the program (lest your master data be overwritten by today's transactions, instead of being merged with today's transactions, etc).

Instead, Multics should provide some mechanism for maintaining a list of data set generations, and for supplying the proper generation as an active function.

Constraints

The mechanism must satisfy the following constraints:

1. It must handle both disk files and tape files.

2. The particular naming convention chosen for the actual files must be specified by the user, since it must be meaningful to her application.

---

3. The active function interface must have the ability to create a new generation or reference an existing generation, depending upon the value of a generation indicator given as an argument. For example, an indicator of 0 would ask for a new generation, 1 would ask for the most current existing generation, 2 would ask for the 2nd most current generation, etc. Thus, the generations take on the nature of a push down stack.

PROPOSED SOLUTION

1. A generation data set will be described by a file, which catalogues the various generations. The file will be unstructured, so it can be printed and edited. It will have the name of the generation data set, with a suffix of gds.

2. Each line in the file will describe a generation of the data set, with most current generations appearing first, olest generations appearing last.

3. A line will consist of an attach description, giving the I/O module, file pathname or tape parameters, etc.

4. The file will be maintained by an active function called generation_data_set (gds) whose operation is described below. Normally, the file will have only r entries in its ACL, to protect its contents from accidental overwriting. Its safety switch will be turned on. The gds active function will attempt to set access for the user while updating the list of generations, and will then restore the ACL to its original state. A cleanup on unit will restore the access in case the update fails or is aborted. The cleanup on unit will also insure that an update is complete, so that the file remains consistent.

Name: generation_data_set, gds

Syntax: [gds gds_path {-control_args}]

Arguments:
gds_path is the pathname of the generation data set catalogue. This pathname identifies the generation data set. A suffix of gds is assumed if not given.

Control argments:
-generate N, -gen N Specify that the Nth oldest generation is to be returned.
-input_description ARGS, -ids ARGS specify that the remaining arguments form an attachment description which identifies a new generation of the data set.

Notes:
If no control arguments are given, gds returns the most recent generation. If only -ids is given, gds adds the specified generation as the newest generation in the catalogue, and returns that generation as its result.

If only -gen is given, then N >= 1. The Nth oldest generation is returned.

If both -gen and -ids are given, then N >= 0. When N >= 1, the Nth oldest generation is returned and the input description given with -ids is ignored. When N = 0, the input description is stored as the newest (0th) generation, and is returned by gds.