SAS® GLOBAL FORUM 2021

Paper 1123-2021

How to Deal with Locked Datasets on UNIX Systems

Kurt Bremser, Allianz Technology Austria

ABSTRACT

This paper presents a method to help in updating datasets that are in use by another process.

INTRODUCTION

Quite often, it happens that we need to create a new version of an existing dataset, but cannot do so, because another user and/or process has the dataset open (in a viewtable window, for instance), and we cannot get the required lock.

Or one might have an issue with a dataset created by another user, which therefore does not grant write permission to others.

I will present a workaround that allows to do this, without requiring dedicated service windows where user access to data is prevented, or other means to make a file writable.

BACKGROUND

THE UNIX FILESYSTEM

A UNIX filesystem consists (roughly) of these parts:

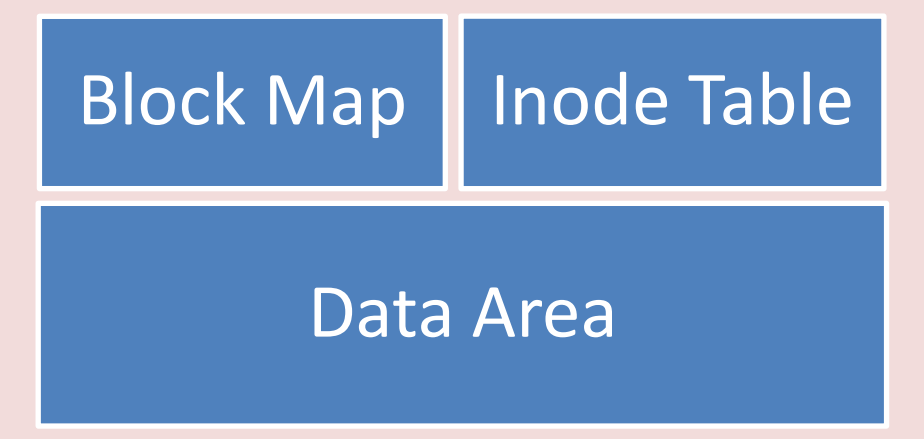

Figure 1: The UNIX Filesystem

- The block map contains information about used and free blocks in the data area.
- The inode table has at least one entry (inode) for each file, where file metadata (owner, timestamps, permissions, size, allocated blocks) is stored. The inodes do *not* contain names.
  Continuation inodes are used if more allocated blocks are needed than can fit into the first inode.
- The data area contains the data blocks.

- Directories are stored like other files. They only contain file names and pointers to the inode table, a directory entry is also called a "link" in UNIX parlance.

When a file is opened, the system searches the directory for the file name, retrieves the inode number, and opens the file by using the inode; as soon as the file handle is established, the directory entry becomes irrelevant.

This implies that a directory entry for a given file can be changed, added or removed while the inode is in use. The method described makes use of this.

WHAT ABOUT WINDOWS?

Windows does not have the concept of an inode table. File metadata is stored in (or closely related to) the directory entries, and a directory entry is locked while a file has an open file handle. This means you cannot rename or remove a file while it is accessed.

HOW IT WORKS

TECHNIQUES USED IN THE CODE

As SAS itself does not allow overwriting a *dataset* while it is in use of another process, we need to deal with the *file* itself in the context of the file system.

The code uses either the external **`rm`** command with the **`-f`** option to remove the directory entry for the dataset, or the FDELETE function.

The external command can be run with %SYSEXEC or with INFILE PIPE. Similarly, the FDELETE function is either used with %SYSFUNC as pure macro code, or as part of a DATA _NULL_ step.
INFILE PIPE allows the retrieval of all system messages, so they can be written to the SAS log.

The **`-f`** option of the external command prevents any request for confirmation coming back from the command (e.g. if one does not have write permission for the file itself; as long as write permission for the directory is granted, the command will work without further requests for confirmation).

THE MACRO ITSELF

```
%macro ds_delete(ds=,ext=Y,safe=N);

/* ds_delete, Version 0.9
   This macro is designed to use the UNIX operating system command rm to
   delete a dataset or view, in order to prevent a "Dataset is locked"
   message. It will automatically determine the physical path to the
   library and the proper filename extension. Depending on the presence
   of a second parameter, it will either use the FILENAME PIPE method to
   run the UNIX command, or the %SYSEXEC macro statement.
*/

%if %length(&ds.) lt 1
%then %do;
  /* issue a usage note */
  %put Usage:;
  %put %nrstr(%ds_delete)(ds=dataset|view[,ext=y|n][,safe=y|n]);
  %put ds=dataset|view .. name of the dataset or view, single level
considered in WORK;
  %put ext= (optional) .. Y will use external commands;
  %put safe=(optional) .. Y will cause macro code to be used (otherwise
data step);
```

```
  %return;
%end;

/* make our macro behave gracefully */
%local
  dsid
  type
  suffix
  pathname
  fname
  command
  rc
;

/* crude check for validity of dataset name */
%if %sysfunc(countw(&ds.,.)) gt 2
%then %do;
  %put Too many levels!;
  %return;
%end;

/* expand single-level dataset name */
%if %sysfunc(countw(&ds.,.)) lt 2 %then %let ds = WORK.&ds.;

/* check if the dataset/view actually exists */
%let dsid = %sysfunc(open(&ds.));
%if &dsid. = 0
%then %do;
  %put Dataset or view does not exist!;
  %return;
%end;

/* determine physical filename suffix */
%let type = %sysfunc(attrc(&dsid.,MTYPE));
%if &type. = VIEW
%then %let suffix = sas7bvew;
%else %let suffix = sas7bdat;

/* get physical name of the directory */
%let pathname = %sysfunc(pathname(%sysfunc(attrc(&dsid.,LIB))));

/* build the complete filename */
%let fname = %lowcase(%sysfunc(attrc(&dsid.,MEM))).&suffix.;

/* external command or FDELETE */
%if %upcase(&ext.) = Y
%then %do;

  /* build the command */
  %let command = rm -f &pathname./&fname.;

  /* run the command, either with %SYSEXEC or FILENAME PIPE */
  %if %upcase(&safe.) = Y
  %then %do;
    /* this makes the code "safe" for use anywhere in code,
       by using only macro statements */
    %sysexec &command.;
  %end;
```

```
  %else %do;
    /* retrieve all responses (including stderr) and write them to the log
*/
    data _null_;
    infile "&command. 2>&1" pipe;
    input;
    put _infile_;
    run;
  %end;

%end;
%else %do;

  %if %upcase(&safe.) = Y
  %then %do;
    /* macro statements */
    %let rc = %sysfunc(filename(fref,&pathname./&fname.));
    %if &rc = 0
    %then %do;
      %let rc = %sysfunc(fdelete(&fref));
      %put &=rc.;
      %let rc = %sysfunc(filename(fref));
    %end;
  %end;
  %else %do;
    /* data step */
    data _null_;
    length fref $8;
    rc = filename(fref,"&pathname/&fname.");
    rc = fdelete(fref);
    put rc=;
    rc = filename(fref);
    run;
  %end;

%end;

%let dsid = %sysfunc(close(&dsid.));

%mend;
```

WHAT ACTUALLY HAPPENS

The macro removes the directory entry; any subsequent step writing the dataset will not find the "old" version, and will not detect the "open" status, so the dataset can be created without problems.

A user who has the dataset open will not notice anything. As long as a file handle for the inode exists, the data itself will remain; as soon as the system kernel realizes that

- the last handle for an inode is closed
- no link (pointing to the inode) exists

the data will be de-allocated and the inode removed from the inode table.

This also means that during the creation of the new dataset, both "old" and "new" can exist in parallel and require space; with large datasets using up a substantial part of a filesystem or multiple repeats of the process where every time someone keeps the old dataset "alive", this can be a problem.

CONCLUSION

The method presented in this paper has been in use in our company for many years.

It allows us to run batch jobs seamlessly during "normal" operation hours, without the need to disrupt services, and without the danger of jobs crashing just because a user forgot to close a viewtable.

It also allows users to overwrite datasets created by others; with the default UNIX umask of 022 this is usually not possible, as only the original owner can overwrite a dataset.
But the macro depends only on write permission for the directory, which you need to have in the first place to create a dataset there.

WHAT IT CANNOT DO

Obviously, you cannot append to an existing dataset; in this case, you need to create a copy of the dataset in your WORK, append there, and then use the macro and move the appended dataset to the target library.

Index files will not be removed, but SAS clears these files on its own when a new dataset with the same name is created. Since the macro is intended to be used immediately before the new dataset is written, orphaned index file will not be an issue.

Working with generational datasets is also not supported.

Feel free to use the code for your purposes; in case you find useful improvements (or detect issues), please comment the SAS Communities article mentioned in the references.

REFERENCES

Website Kurt Bremser at SAS Communities. 2020. "Avoiding the dreaded "Dataset is locked" ERROR". https://communities.sas.com/t5/SAS-Communities-Library/Avoiding-the-dreaded-quot-Dataset-is-locked-quot-ERROR/ta-p/653210.

ACKNOWLEDGMENTS

A big thanks to the SAS Communities member who first introduced me to the "FILENAME PIPE" method to run external commands; alas, it's been such a long time that I can't remember who she/he was.

RECOMMENDED READING

UNIX File System (https://en.wikipedia.org/wiki/Unix_File_System)

inode (https://en.wikipedia.org/wiki/Inode)

Following the discussions on the SAS Communities is always highly recommended, for beginners and advanced SAS coders alike.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kurt Bremser
https://communities.sas.com/t5/user/viewprofilepage/user-id/11562