TO: MSPM Distribution
FROM: K. J. Martin
SUBJECT: BY.2.11
DATE: 06/20/68

The write-up of working_segs is being re-issued to reflect slight changes in error handling.

Published: 06/20/68
(Supersedes: BY.2.11, 05/28/68)

Identification

working_segs
E. W. Meyer, Jr., S. L. Rosenbaum

Purpose

There are several types of programs, notably translators, which need to create and manipulate temporary segments in the process directory and to later move them into another directory. "working_segs" is designed to allow the command writer to do this with a minimum amount of coding effort.

Discussion

Two entries exist within working_segs: $init and $finish. $init creates the desired segments in the process directory, and $finish moves them into the target directory. Prior to calling working_segs$init, an array (each element of which constitutes information for one segment) must be filled in with the name suffix and maximum length of each segment to be created. working_segs$init creates each segment in the process directory with the indicated maximum length and an entry consisting of the concatenation of a common unique character string with the supplied name suffix, and places a pointer to each segment in the proper position of the corresponding array element. It sets the copy and relate switches off and enables the modes "read", "write", and "append".

Prior to the call to working_segs$finish each element of the array must be filled in with the permanent settings of the bit count, maximum length, attributes, and modes. $finish moves the segments to the specified directory, replacing the unique character string first component of the name with the specified permanent component, and deletes the process directory branches. If it cannot move one segment (see discussion of status below), it sets the appropriate value for status and continues with the next segment. Note that the segment suffixes and the unique name stored in the segment information aggregate by $init must not be altered between the calls to $init and the call to $finish; otherwise, $finish will be unable to find the segments that it is to move. Because all the static information is stored in the supplied structure, working_segs can operate on several sets of segments simultaneously (i.e., asynchronous calls to $init and $finish are allowed).

Usage

```
call working_segs$init (segs);

call working_segs$finish (segs, target_dir, perm_name)

dcl  target_dir char(*),                    /*target directory
                                              pathname*/

perm_name char(*);                          /*permanent name prefix
                                              to replace unique string
                                              in the target directory
                                              branches*/


dcl  1 segs,
       2 uname char (15),
       2 array(*),
         3 status fixed bin (17),
         3 replace bit (1),
         3 base_ptr ptr,
         3 vacant bit (1),
         3 max_length fixed bin (9),
         3 bit_count fixed bin (24),
         3 trewa char (5),
         3 copy_relate bit (2),
         3 count fixed bin (17),
         3 suffix char (32);
```

uname - unique character string created by $init.

status - indicates the success of creating (or moving) this segment. segs.array(j).status takes one of the following values upon return from $finish.

=0 entry (j) successfully created (or moved)

=1 entry of same name as the segment to be moved currently exists in the target directory and the value of replace (see below) directed that $finish should not attempt to replace the segment by the working segment.

=other - File system error code. The basic file system could not create the working segment (or could not move it to the target directory). The user can call check_fs_errcode (section BY.2.02) to determine precisely what error occurred.

replace - indicates the action to be taken if the current entry (perm_name || suffix) already exists. If replace = "0",b, working_segs returns segs.array(j).status =1. If replace="1",b, working_segs attempts to truncate the segment and replace it by the working segment. (Note: The write attribute must be on for a segment in order to truncate it).

base_ptr - pointer to created segment - filled in by working_segs$init.

vacant - If "1"b, $init ignores this array element; $finish tries to delete the working segment from the process directory if it exists, (i.e., the switch was "0"b in the call to $init).

max_length - maximum length of created segment in units of 1024 words - required by $init and $finish (need not be the same value for both calls).

bit_count - length of the segment in bits-required by $finish to set the bit count in the target directory branch.

trewa - specifies final settings of the access modes for this user. If the "append" mode is to be set off, the maximum length for this segment will be set to the minimum needed to completely encompass the current length; otherwise max_length specifies its value. If trewa is null, a default is taken depending on whether or not the branch already exists: if it exists, its current mode is used; if it does not already exist, read, write and append modes are used.

copy_relate - specifies final settings for the copy and relate switches, respectively. Note: if a branch of the same name exists in the target directory and if all modes are off, i.e., trewa equals five spaces, then the currently existing mode and switch settings will be used, i.e., the settings for copy_relate will not be used - required by $finish.

count - number of characters in supplied name suffix - required by $finish.

suffix - entry name suffix, 32 characters in length or less. The user must supply the "." in the string if it is a component suffix. The length must be stored in "count" - required by $init and $finish.

Implementation

working_segs$init takes the following action for each array element:

1. If segs.array(j).vacant="1"b this element is ignored.

2. smm$set_name_status is called to create the segment of maximum length = 1024 * segs.array(j).max_length and return its base pointer.

working_segs$finish does the following for each array element:

1. If segs.array(j).vacant="1"b, then call smm$terminate to get rid of the temporary segment in the process directory. Go to the next array element.

2. Otherwise, call move_file_ to move the segment from the process directory to the target directory. If the segment cannot be moved, working_segs checks for one of the following error codes:

   a. No entry exists in the target directory. Retrieve the final mode and switch settings from segs.array(j).trewa and segs.array(j).copy_relate. Call append_branch to create a zero length segment with proper mode, switch settings and maximum length. Try to move the segment again.

   b. A non-zero length entry exists in the target directory. If segs.array(j).replace="0"b then set segs.array(j). status=2 and go to next array element. If segs.array (j).trewa equals the null string then call entry_status $switches to retrieve current mode and switch settings. Call truncate_seg to reduce the current segment to zero-length. Try to move the segment again.

3. If the segment is not successfully moved, set segs.array(j). status to the file system error code returned by move_file and go to next array element.

4. When the segment has been successfully moved, call set_count$bits to set the branch's bit count. If entry previously existed, call set_max_length to set the branch's maximum length; call set_copy to set the branch's copy switch; call set_relate to set the branch's relate switch; call mode$set to set the branch's access modes for the current user. Call smm$set_del_sw and smm$terminate to delete the working segment from the process directory.