Published: 10/02/68

Identification

CTSS to Multics Transition Aid
adjust
Charles Garman

Purpose

The adjust command is designed for use by library maintenance personnel in moving ASCII files from CTSS to Multics. It is also useful as a debugging tool in "picking up the pieces" should a command which creates an ASCII file fail to set the bit-count in the branch for the segment (see Implementation, below).

Discussion

In Multics, the standard means for identifying the real amount of data in a segment is to set the bit-count in the branch for the segment to the proper multiple of the element size; in the specific case of ASCII segments (EPL source, etc) this value is 9 times the number of characters in the segment. (All system commands which write ASCII segments set this value, all system commands which attempt to read ASCII segments read characters up to this value).

In CTSS, which maintains file lengths to the closest (higher) word boundary, the ASCII ETX character (octal value 003) was chosen as an interim solution for marking the end of a file whose last character was not in the 4th character position of the final word. (e.g., 073 012 003 000(8) represents the last word of a file which contains ";<NL>" as its last two characters, with the ETX marking the end, and the 000, ASCII NUL, used as a padding character).

The current means of inputting segments in Multics, from Multics System Tapes during a Multics Bootload or the Tape Daemon, both set the bit-count, but generally only to the nearest (higher) multiple of 36, the number of bits per word.

The adjust command remedies the situation by looking at the segment specified, starting from the character position derived from the bit-count, searching backwards until it finds the first non-NUL, non-ETX character. (See refinements under Implementation). It then sets to NUL all trailing characters in the final word, truncates the segment beginning at the next higher word, and, if this operation should result in shortening the segment by 1 1024-word block or more, it also resets the maxlength. It then re-calculates the bit-count, and sets the value in the branch for the segment.

Usage

```
{adjust
 adjust$test
 adjust$block
 adjust$test_block}      name
```

name is the name of the segment (to be found in the working directory if no ">" characters are in the string).

adjust merely executes as described.

adjust$test looks at the segment, but doesn't do anything, it merely prints diagnostic messages indicating what it would have done had the normal entry been invoked.

adjust$block ignores the given bit-count and uses the current-length in 1024-word blocks to calculate its initial position

adjust$test_block combines the extra features of $test and $block.

Implementation

For the various entry ponts, switches are set to control later program flow.

A pointer to the segment is obtained by calling smm$initiate, and the current length, maximum length and bit-count are obtained by calling entry_status$lengths.

If the bit-count was zero, or one of "block" entries was called, the number of characters is calculated from the current length, otherwise the bit-count is divided by 9 to get the character count (if the bit-count was not originally a multiple of 36, adjust comments that it has already been invoked for the segment but starts at the last character of the word). If both the bit-count and the current length are zero, it checks starting from the maximum length.

Once the initial character position has been determined the searching procedure starts:

1. once every 4 characters, the word index is calculated and tests are made for the entire word being binary zero, or for high order bits in any ASCII character position in the word; if either case is true, all characters in the word are skipped, and in the case of high-order bits, a switch is set for later interrogation.

2. for each character, if it is not NUL, ETX, or NL, good information has been found, and searching stops. If NUL or ETX characters were found, the character is skipped and searching continues.

3. if a NL character was found, and if the preceding character is not NUL, the NL character is the final character of the segment, and searching stops. Otherwise, both the NL and NUL characters are skipped over and searching continues (back to step 1). (This part of the algorithm deals with an unfortunate interaction between the QED and ASCII-ARCHIVE commands in CTSS, which can result in extending the file with an indefinite number of extra lines containing only NUL and NL characters.)

Once the searching process has terminated (either in steps 2 or 3, or by searching back to the beginning of the segment without finding a non-NUL, non-ETX character), several cleanup steps take place, (either for real, or with diagnostic messages if the "test" entries were called):

4. characters in the final word, beyond the last usable character, are replaced by the ASCII NUL character.

5. if words containing high-order bits were encountered (step 1) a diagnostic is printed.

6. the offsets of the first word after both the starting character and the final character are calculated; if the second value is less than the first, the segment is truncated at that word (i.e., the word after the last data character).

7. if the new current length is less than the original maximum length, the maximum length is reset to the (new) current length.

8. the new bit-count is computed from the character count and set back in the branch.

9. the segment is terminated.

Errors

If errors are reflected from the basic file system, execution stops and control is returned to adjust's caller (e.g., to command level in the listener).