MULTICS TECHNICAL BULLETIN MTB - 021

TO: Distribution

FROM: Ross E. Klinger

DATE: November 30, 1973

SUBJECT: New version of the sort_file command, and a new subroutine, sort_file_ .

Attached are the MPM write-ups and MCR draft for the proposed command sort_file, and its associated subroutine, sort_file_ . The new procedures provide a number of capabilities which were not implemented by the current sort facility, as well as an appreciable increase in speed of operation. The eventual implementation of a multi-segment file capacity should require no changes to the user interface.

If you would like to try these procedures, the currently reside in >udd>pdo>Klinger>public . In any case, I would appreciate your comments and suggestions. Send written comments to:

Ross E. Klinger
MIT Room 39-464

or mail comments to: Klinger.PDO

Multics Project internal working documentation. Not to be reproduced or distributed outside the Multics Project.

ENHANCED FLEXIBILITY

The new sort_file command provides the following facilities not implemented by the current command:

1) specification of an output segment for the sorted results

2) specification of the line delimiter

3) specification of descending sort order

4) ability to sort blocks of lines

5) ability to sort on multiple fields within lines or blocks of lines

ENHANCED EFFICIENCY

Sorting a large file (24 disk records), the new command is 35% faster than the current command for the (only) case which the current command can process; i.e., ascending sort on an entire line, sorted results to replace the original contents of the file. In seconds, this would be on the order of 50 vs. 85.

Sorting a small file (1 disk record), the new command is 50% slower than the current command, for the case which the current command can process. In seconds, 1.3 vs. .9 . The additional processing time stems from the necessity to bypass portions of code on each sort pass which are not used by the simple case. As the size of the file increases, this additional processing time becomes negligible. The flexibility/efficiency tradeoff has been minimized.

SUBROUTINE INTERFACE

Currently, users desiring to use the sort_file facility as a subroutine call were forced to use the command interface. The development of a subroutine interface eliminates this source of inefficiency. Furthermore, an entry point has been provided so that small quantities of data within a segment may be sorted, whereas previously it was only possible to sort an entire segment. In combination with the implementation of delimiter and field specifications, the subroutine interface can be used to sort almost anything, anywhere.

Name: sort_file, sf

The sort_file command orders the lines of an ASCII file according to the ASCII collating sequence. The sort is stable.

Usage sort_file pathname -control_args-

1) pathname — specifies the pathname of the input file to be sorted. pathname may be either an absolute or relative pathname.

2) control_args — may be chosen from the following list of control arguments:

-output_file path
-of path — specifies that the sorted units are to be placed in a segment whose pathname is path. path may be either an absolute or relative pathname. The use of this control argument is incompatible with the use of the -replace control argument. (See -replace , below)

-replace
-rp — specifies that the sorted units are to replace the original contents of the input file. This is the normal mode of operation. The default is equivalent to an explicit specification of -replace . (See -output_file , above)

-delimiter xyz
-dm xyz — specifies the character string which, when concatenated with a newline character, will be considered as the line delimiter. xyz may be any sequence of ASCII characters. The default is a single newline character. (See Examples)

-block n
-bk n — specifies the sort unit to be a block of n lines. n must be a positive integer. The default is n = 1. (See Examples)

-descending
-dsc — specifies the sort to be in descending order, according to the ASCII collating sequence. The use of this control argument

c Copyright 1973, Massachusetts Institute of Technology and Honeywell Information Systems Inc.

is incompatible with the use of the -ascending control argument. (See -ascending, below)

-ascending
-asc

specifies the sort to be in ascending order according to the ASCII collating sequence. This is the normal mode of operation. The default is equivalent to an explicit specification of -ascending . (See -descending , above)

-field S1 L1 S2 L2 .... Sn Ln
-fl S1 L1 S2 L2 .... Sn Ln

specifies the field (or fields) within a sort unit which will be considered when sorting. A sort field is defined by a pair of field specifications, S and L. S is the start position of the field, specified in characters relative to the first character of a sort unit. L is the length of the field, in characters. Both S and L must be positive integers. The first pair of field specifications, S1 L1, defines the primary sort field; the second pair, S2 L2, defines the secondary sort field; and so forth.... The use of this control argument is incompatible with the use of the -all control argument. (See -all , below) (See Notes)

-all
-a

specifies the primary (and only) sort field to be the entire sort unit; i.e., the entire sort unit is considered when sorting. This is the normal mode of operation. The default is equivalent to an explicit specification of -all . (See -field , above)

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Notes

The start position of a sort field is calculated relative to the first character of a sort unit. If the blocking factor is n = 1, then the start position is calculated relative to the first character of a line. If the blocking factor is n > 1, the start position is calculated relative to the first character of the first line of a block. When calculating field specifications within a sort unit of n > 1 lines (blocking factor n > 1), line delimiters internal to the sort unit should not be considered. (See Examples)

Sort fields/units of unequal length are compared by assuming the shorter field/unit to be padded on the right with blanks, immediately following the rightmost character. The line delimiter is never considered when padding. (See Examples)

If characters are detected in the input file following the final delimited sort unit, they will be ignored for the purposes of sorting, but will appear in the sorted output immediately following the final delimited sort unit. An error message will specify the location of the first non-delimited character.

Sorting is performed in two steps. The radix exchange algorithm is employed to sort on the first n (where n <= 16) characters of the sort field/unit. This is followed by a sort on the entire sort field/unit using a modification of the Shell algorithm. The sort is stable.

The file is sorted using temporary segments in the process directory. If the -output_file option is specified, and path is the pathname of a pre-existing segment, its contents will be destroyed upon beginning the sort. If the sorted results were to replace the original contents of the input file, that replacement does not occur until the last possible moment.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

Examples

Given the following input:

ABCDEFGHXYnlABCDEFXYnlABCDEFGHIJXYnlABCXYnl

where: nl stands for the newline character
ø stands for the blank character

these control arguments	will result in these sort units	being sorted on these sort fields	
-dm XY	ABCDEFGH	ABCDEFGHøø	
	ABCDEF	ABCDEFøøøø	
	ABCDEFGHIJ	ABCDEFGHIJ	
	ABC	ABCøøøøøøøø	
-bk 2	ABCDEFGHABCDEF	ABCDEFGHABCDEF	
-dm XY	ABCDEFGHIJABC	ABCDEFGHIJABCø	
-fl 6 4	ABCDEFGHXY	FGHX	
	ABCDEFXY	FXYø	
	ABCDEFGHIJXY	FGHI	
	ABCXY	øøøø	
		primary field	secondary field
-fl 1 4 7 2	ABCDEFGHXY	ABCD	GH
	ABCDEFXY	ABCD	XY
	ABCDEFGHIJXY	ABCD	GH
	ABCXY	ABCX	øø
-dm Y	ABCDEFGHXABCDEFX	FGHX	DE
-bk 2	ABCDEFGHIJXABCX	FGHI	DE
-fl 6 4 4 2			

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (End)

Name: sort_file_

The sort_file_ subroutine orders the lines of an ASCII file according to the ASCII collating sequence. The sort is stable.

Usage

```
dcl sort_file_ entry (char(*), char(*), char(*), char(*),
    char(*), bit(1), fixed bin(35), ptr, fixed bin(35),
    fixed bin(35));

call sort_file_ (in_dir, in_ent, out_dir, out_ent, delim,
     ad, block, fptr, cad, code);
```

1) in_dir — is the directory name of the input file to be sorted. (Input)

2) in_ent — is the entry name of the input file. (Input)

3) out_dir — is the directory name of the output file in which the sorted results will be placed. See Notes. (Input)

4) out_ent — is the entry name of the output file. See Notes (Input)

5) delim — is the ASCII character string considered to be the line delimiter. See Notes. (Input)

6) ad — is the ascending/descending sort order bit.

= "0"b for descending order
= "1"b for ascending order

(Input)

7) block — is the number of lines per sort unit. (Input)

8) fptr — is a pointer to the user declared field specifications structure. See Notes. (Input)

9) cad — is the position, in characters, relative to the first character of the input file, of the

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

first undelimited character in the file. See Notes. (Output)

10) code — is a standard Multics error code. See Notes. (Output)

Entry: sort_file_$ptr

This entry point is used to sort data within a previously initiated segment, and to place the sorted results in a previously initiated segment.

Usage

```
dcl sort_file_$ptr entry (ptr, fixed bin(24), ptr, char(*),
    bit(1), fixed bin(35), ptr, fixed bin(35), fixed bin
    (35));

call sort_file_$ptr (in_ptr, bc, out_ptr, delim, ad, block,
     fptr, cad, code);
```

1) in_ptr — is a pointer to the data to be sorted. (Input)

2) bc — is the bit count of the data to be sorted. The value of bc must satisfy the condition mod(bc, 9) = 0. (Input)

3) out_ptr — is a pointer to the area where the sorted results are to be placed. The output area may either completely or partially overlay the input area. (Input)

4) delim

.
.
. — are defined as for sort_file_ .
.

9) code

Notes

1) See the MPM write-up of the sort_file command for a detailed explanation of sort units, field specifications, algorithms used, etc.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

2) If the sorted results are to replace the original contents of the input file, out_dir and out_ent must be specified as null ("") character strings. out_dir and out_ent must never specify the same file as in_dir and in_ent .

3) delim is the actual line delimiter. sort_file_ does not concatenate a newline character to the delimiter string as does the sort_file command. If the newline character is to appear in the delimiter string, in either a terminal or medial position, it must be explicitly included in the string.

4) The field specifications structure must be declared as follows:

```
dcl 1 field_specs aligned,
      2 n fixed bin,
      2 fields (n),
        3 start fixed bin (35),
        3 length fixed bin (35);
```

where: n is the number of sort fields
n is the value of n
start is the start position of the field
length is the length of the field

fields(1) is the primary sort field, fields(2) is the secondary sort field, and so forth....

5) cad contains a valid value only if the error code is error_table_$chars_after_delim .

6) sort_file_ will return one of the following error codes:

0 - normal return

$chars_after_delim - characters were found after the final delimited sort unit. The undelimited characters were placed following the final delimited unit in the sorted results, and cad contains the location of the first undelimited character. The sort was completed.

$no_delimiter - no delimiters were found in the input file. The file was not sorted.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc.

$no_makeknown - the input file could not be initiated. The file was not sorted.

any error code returned by hcs_$make_seg, hcs_$truncate_seg, or hcs_$set_bc_seg . The file was not sorted.

c Copyright 1973, Massachusetts Institute of Technology
and Honeywell Information Systems Inc. (End)

Ver. 2
730919

MULTICS CHANGE REQUEST

MCR _____

TITLE: replace current sort_file command and add a new sort_file_ subroutine

STATUS: Written 11/23/73; Approved ____; Rejected ____; Postponed ____; Withdrawn ____; Expires ____

DATE: _____

AUTHOR: Ross E. Klinger

SOURCE: (if external; e.g., "User", "Marketing")
MIT - PDO

CLASSIFICATION:
- [ ] Incompatible Change
- [ ] Extension
- [ ] Restriction
- [x] Performance Improvement
- [ ] Reliability Improvement

JUSTIFICATION:
- [ ] Marketing Requirement
- [ ] Conformance to Standard
- [ ] Increased Consistency
- [ ] Simplification
- [x] Generalization
- [ ] Bug Fix

Replaced by proposal MCR _____

Implemented in System _____

Objections/Comments:

Use these headings: REASONS, SUMMARY, IMPLICATIONS, and optionally DETAILED PROPOSAL

REASONS:

1) to provide a more flexible sort facility
2) to provide a more efficient sort facility
3) to provide a subroutine interface which can sort an entire segment, or portion thereof

SUMMARY: see attached sheets

IMPLICATIONS:

1) replace the current sort_file command in >system_library_standard>bound_misc_commands_
2) revise the MPM write-up on sort_file
3) add the subroutine sort_file_ to the appropriate system library
4) include the write-up on sort_file_ in the MPM
5) add two new codes to error_table_

DETAILED PROPOSAL: see attached sheets