M.I.T. Laboratory for Computer Science
Computer Systems Research Division

February 3, 1977
Request for Comments No. 134

Measurements of Sharing in Multics.

By Warren Montgomery

The following is a report on an attempt to measure the extent of sharing in the M.I.T. Multics system. The report is primarily concerned with types of sharing that are difficult to provide in a distributed system, such as sharing of writeable objects and sharing of main memory pages. Several measures of sharing are reported and analyzed.

1) Motivation:

One of the primary design goals for the Multics system was to provide extensive facilities for controlled sharing. To help achieve this goal, the multics system allows user processes to share physical memory pages. Thus Multics allows for a great deal of interaction among user processes through shared pages. It is unclear, however, just how much the average user depends on such sharing in his use of Multics.

It would be very difficult to provide this level of sharing in a distributed system. It is interesting, therefore, to measure the extent of sharing in a system such as Multics. Such measurements would be of great value in determining the strategies used to provide sharing in a distributed system.

2) Definitions:

Before we can attempt to measure sharing, we must settle on a precise definition of what it is we should be measuring. The definition of sharing that follows is an attempt to capture the aspects of shared objects that may be difficult to provide in a distributed system.

> An object is said to be shared in a time window t if it is used by at least 2 different processes within the time t.

There are several terms in this definition that require further clarification. The term object refers to a logical unit of information. We want to measure the sharing of different types of objects separately, because different types of objects are used in different ways, some of which may be more difficult to provide in a distributed system.

There are many ways in which objects can be used. In this report, I will distinguish two kinds of uses: reads, which obtain information from an object but do not modify its contents, and writes, which modify an object. An object is read-shared in the time window t if it is shared in t and was not written in tt. Similarly, a write-shared object is one that is shared and written. I chose to make the distinction between read-sharing and write-sharing because read-sharing is in general easier to provide than write-sharing. We can provide "sharing" of an object that is only read by giving each process that reads the object a private copy. The processes cannot detect the difference between such an implementation and one in which all processes share a common copy. If a shared object is written, however, the information obtained by subsequent reads in any process must reflect the modifications made by the write.

Notice that I have chosen to define sharing in terms of the use of objects by processes instead of the use of objects by users (people). This choice is made because I am interested in determining how much sharing must be provided for by the hardware and the operating system, which deal with processes. In the Multics system, each user has control of a single process, so a measurement of sharing based on references by users would not be much different from one based on references by processes.

The time window in the definition is a very important parameter. This window is an indication of the communication bandwidth needed to provide sharing. A much higher bandwidth is needed to provide sharing in a small time window than is needed to provide sharing in a large time window.

We can now ask ourselves what is the proper way to parameterize sharing. Different sets of parameters are useful for different purposes. In this report, we are most interested in determining how easily sharing can be

provided in a distributed system. The remainder of this section describes several parameters that I feel would be most useful in describing sharing in a distributed system.

One interesting parameter to measure is the percentage of objects of each type supported by the system being measured that are shared at some point in their lifetime. This parameter could be used to determine whether or not to provide for the sharing of an object at the time that it is created. If the measurement shows that very few objects of a particular type are ever shared, then sharing should not be considered heavily in the design of an implementation of objects of that type.

Similarly, we can ask what percentage of objects are ever shared by concurrent processes. An object can be shared by our definition without ever being used by concurrent processes. Such a situation occurs when several processes use an object, but their lifespans do not overlap. Sharing of objects by concurrent processes is much more difficult to provide in a distributed environment than sharing of an object by processes whose lifespans do not overlap.

Another interesting parameter would be the percentage of objects referenced in the last t1 seconds that were also shared in the last t2 seconds. This provides some indication of the communication bandwidth needed to provide sharing. The time interval t2 is the time window described in the definition of sharing, while the time interval t1 selects the objects of interest. (A small t1 selects only recently used objects, while a large enough t1 selects all objects.) For a given interval t1, we would expect this parameter to increase as t2 is increased. The value of this parameter for t1 and t2 both infinite is the percentage of all objects that are shared at some point in their lifetime.

A very important parameter describing sharing is the percentage of all references to objects that reference shared objects. The parameter described in the preceding paragraph can be used to get an approximation to this parameter. If t1 is chosen very small (on the order of the time required to reference an object), the percentage of objects referenced in the last t1 seconds that were shared in the last t2 seconds approximates the percentage of all references to objects that referenced objects shared in the last t2 seconds. If in addition we can estimate the amount of communication needed to support one average reference to a shared object, we can use this parameter to estimate the communication bandwidth needed for sharing.

In addition to measuring the proportion of objects that are shared and the proportion of references to objects that reference shared objects, it is interesting to know how many processes are sharing the average object. If each process must be given a private copy of a shared object, then this parameter gives an estimate of the storage required to hold the copies. Thus it would be desirable to see the distribution of the number of processes sharing an object in the time window t2 for all objects referenced in the last t1 seconds.

3) Measurement Techniques.

The parameters described above would provide a good estimate of the storage capacity and communication bandwidth needed to provide sharing in a distributed system. Unfortunately, the Multics system does not provide the information necessary to directly measure these parameters: they must be indirectly measured or estimated. This section describes the measurements that can be easily made, and how these measurements relate to the desired sharing measurements described above.

I studied sharing by observing "snapshots" of the M.I.T. Multics system. Some of the snapshots that I used were obtained from the system dumps generated at the time of a crash, while others were obtained by copying the relevant data bases from a running system. The data obtained from these two methods appear to be consistent.

The results presented in this report come from a snapshot of the M.I.T. Multics system taken in January of 1977. This snapshot was typical of all of those examined, and the figures reported varied very little from snapshot to snapshot. The system was loaded with about 50 users, many of whom were involved in program development, and thus were editing, compiling, or debugging programs. Another large class of users was involved in document preparation.

I studied the sharing of three different types of objects provided by Multics: Segments, Directories, and Pages. (1) The Active Segment Table (AST) of the Multics system contains data on approximately 1000 of the most recently used segments and directories. My sharing measurements were derived from this data. (2)

The segments and directories in the AST are known as active segments. The AST is managed by an algorithm that keeps the most recently referenced segments active. A segment must have been referenced within the last 3 minutes to remain in the AST. (The exact time depends on the size of the

---

(1) There is some confusion over the meaning of the term "segment" in Multics. I will use this term to denote a file system segment, or non-directory segment. The word segment is sometimes used to include directories as well. When I intend to include directories, I will say so explicitly.

(2) The encacheability algorithm was investigated as another potential source of sharing data. Appendix A of this report contains a discussion of that algorithm, and its relevance to sharing measurements.

segment and on the system load). Directories remain active as long as they have descendants that are active.

For each active segment or directory, Multics maintains a list of all of the processes that currently have that segment or directory in their address space. In making my sharing measurements, I counted all segments and directories that appeared in the address space of two or more processes at the time of a snapshot as shared. The correspondence between this definition of sharing and that given above is somewhat unclear, as the time interval of sharing cannot be precisely determined. This time window corresponds roughly to the average lifetime of an object in the address space of a process. This average was about 30 minutes for the snapshots measured. This lifetime varies dramatically, however, depending on the object and the process, so that the correspondence is not exact.

The main memory and the paging device are managed by algorithms that keep the most recently referenced pages in core, and the next most recently referenced pages on the paging device. A page must have been referenced in the last 1 second to remain in main memory (core), and in the last 4 minutes to remain on the paging device. Again, the exact times vary. Thus we can examine the page tables of active segments and directories to determine which segments or directories were read or modified in the last 1 second, or in the last 4 minutes.

There is no information available as to which processes are using a particular page, but by examining the AST entry for the segment or directory that contains that page, we can determine if it is at least possible that that page has been recently shared. Thus we can obtain estimates of such sharing, by counting a page that appeared in at least 2 address spaces at the time of the snapshot as shared.

The amount of write-sharing can be estimated by examining the access that processes have for shared segments. In the Multics system, a process cannot indicate whether or not it intends to modify a segment when it brings that segment into its address space. Each process is therefore given all access rights to which it is entitled for each segment that that process uses. The access that a process has to an object is the only indication that Multics provides as to whether that process has written or will write that object. Because not all processes that have write access to an object actually write that object, the amount of writing is overestimated. Thus the measurements of write-sharing reported below are upper bounds.

From this data, we can arrive at approximations to some of the sharing measurements described in the last section. These approximations are reported in the next section of this report.

4) Results.

This section presents the results of the sharing measurements discussed above. These results are analyzed to determine the causes for any unexpected results, and to determine the implications of all results on the implementation of a distributed Multics-like system.

4.1) The Sharing of Segments and Directories.

Table 1 shows the numbers and percentages of the active segments and directories that were shared at the time of the snapshot. The table shows that about 54% of the active directories were shared, while only 11% of the segments were shared.

Table 1

Sharing of Active Segments and Directories.

	Unshared	Shared
Segments	555 (89%)	70 (11%)
Directories	138 (46%)	165 (54%)

Closer examination of the data used to obtain these results revealed several reasons why a higher proportion of the directories were shared. One reason is that directories stay in a processes address space longer than do segments, because a directory remains in a process's address space as long as any object inferior to that directory remains in the address space of that process. Directories also stay in the AST longer for the same reason.

A second reason that the percentage of shared directories is so large is that all of the process directories appear as shared, but in fact most have not been recently shared. The Initializer process, which creates all other

processes, never terminates process directories. Therefore each process directory appears as shared by its process and the Initializer. If we count process directories as unshared, the percentage of shared directories drops to 43%.

A third reason for the larger proportion of shared directories is that our measurement of sharing will count a directory as shared if two different processes have referenced that directory or any of its inferiors since the directory entered the AST. Thus a directory is shared if any of its inferiors are shared (or if two of its inferiors are used by different processes.) Therefore we would expect a high proportion of shared directories.

Although a higher proportion of directories than of segments are shared, it is unlikely that shared directories would cause as much trouble in a distributed system as as would shared segments. Directories are generally smaller than segments. From my data, the average size of an active segment was 8.6 pages, while the average size of an active directory was 3.5 pages. The cost of maintaining duplicate copies, or of communicating updates, should therefore be smaller for directories than for segments. Directories are also less frequently referenced than segments.

Another point to consider is that directories are referenced through calls to the supervisor, and processes are prepared for delay in the execution of such calls. Thus additional delay due to communication in a network would not be a severe problem. Processes reference segments directly, through machine instructions, and are not prepared for long delays. Furthermore, the supervisor synchronizes references to shared directories, while processes rely on sharing the same copy of a segment to synchronize their references to it.

For these reasons, many of the measurements made for this report were made only on segments.

4.2 Read-Sharing and Write-Sharing of Segments.

Table 2 gives the numbers and percentages of active segments that were read-shared or write-shared at the time of the snapshot.

Table 2

Read-Sharing and Write-Sharing of Segments

Read-Shared	61	(9.7%)
Write-Shared	9	(1.4%)

These figures suggest that very little write-sharing of segments takes place. Most of the write-shared segments were actually the representation of some extended type object, such as a Message Segment, or a Mailbox. Such segments are referenced by a small set of programs that carefully synchronize modifications. As noted previously, these estimates of write-sharing are probably above the actual amount of write-sharing in the system.

4.3 The Effect of System Processes and Segments.

The measurements of sharing above include all of the processes, segments, and directories in the Multics System. We would like to be able to examine sharing of objects by user controlled processes separate from that by system processes. Sharing due to system processes may be more easily provided for in a distributed system, and thus the sharing that is solely due to the interaction of user controlled processes may be of greater interest. It is also interesting to examine the sharing of user-created objects separate from that of system objects such as program libraries and bound procedure segments. Table 3 gives the results of performing some of the measurements presented in tables 1 and 2 on the same system with the effect of system processes removed, and on that system with both the system processes and system segments and

directories removed. The table gives the amounts and percentages of read-sharing and write-sharing of active segments, and of all sharing of active directories. The table gives two sets of figures, one for each of the above mentioned systems.

Table 3

The Effect of System Processes and Objects on Sharing Measurements

	User Processes and All Objects	User Processes and User Objects.
Read Sharing of Segments	56 (10.4%)	5 (1.1%)
Write Sharing of Segments	3 (0.6%)	1 (0.2%)
Sharing of Directories	62 (22%)	44 (18%)

These figures show that the system processes are responsible for a substantial proportion of the sharing of directories, and much of the write sharing of segments. The address space of the Initializer process contains a large number of directories that only appear in the address space of one other process. Thus removing the effects of the Initializer process greatly decreases the amount of directory sharing. As noted before, there are very few user programs that share writeable segments, so it is not surprising that the system processes account for most of the write-sharing.

The figures above also show that most of the read-shared segments are system segments. This is not surprising, as the average Multics user process uses a great many system programs that are contained in system segments. These segments account for most of the read-sharing.

4.4) The Average Number of Processes that Share an Object.

The figures presented so far suggest that sharing of segments may not be very important in Multics. The percentages of shared segments seen here suggest that it is relatively rare that a particular segment is shared. It is possible however that the shared segments are much more heavily used than the unshared segments and are therefore very important. One way to try to measure such an effect is to measure the number of processes that share each active segment and directory, and to compute the average. Table 4 gives this number for segments and directories for three different cases: All processes referencing all active segments and directories, User processes referencing all active segments and directories, and User processes referencing active user segments and directories.

Table 4

The Number of Processes that Share an Average Segment or Directory.

	Segments	Directories
All Processes, All Objects	2.5	3.8
User Processes, All Objects	2.5	3.1
User Processes, User Objects	1.1	1.9

The figures show that although few segments and directories are shared, these few are enormously popular. Again, much of the sharing is due to the system segments. Figure 1 shows a histogram of the number of processes shared by exactly N processes as N varies between 1 and 50. Notice that most of the segments are unshared or shared by only 2 processes, while a few are shared by most of the processes on the system. These few are the bound procedure segments that contain the most popular system programs.

Figure 1

The Number of Segments Shared by Exactly N Processes

```
 N

 1  (555)  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
 2  ( 19)  XXXXXXXXXXXXXXXXXXX
 3  (  9)  XXXXXXXXX
 4  (  6)  XXXXXX
 5  (  3)  XXX
 6  (  1)  X
 7  (  4)  XXXX
 8  (  3)  XXX
 9  (  2)  XX
10  (  1)  X
11  (  1)  X
12  (  0)
13  (  2)  XX
14  (  0)
15  (  0)
16  (  0)
17  (  0)
18  (  0)
19  (  1)  X
20  (  0)
21  (  0)
22  (  0)
23  (  0)
24  (  0)
25  (  2)  XX
26  (  0)
27  (  0)
28  (  0)
29  (  0)
30  (  2)  XX
31  (  0)
32  (  0)
33  (  0)
34  (  0)
35  (  0)
36  (  0)
37  (  0)
38  (  2)  XX
39  (  1)  X
40  (  0)
41  (  1)  X
42  (  1)  X
43  (  0)
44  (  1)  X
45+ (  8)  XXXXXXXX
```

4.5) Sharing of Pages.

As noted earlier, the Multics system does not provide as much information on the sharing of pages as it does on the sharing of segments and directories. We can, however, make an estimate of the sharing of pages by examining the information available for the segments and directories to which the pages of interest belong. By examining the pages in main memory, and the pages on the paging device, we can derive estimates of the sharing of pages referenced in the last 1 second, and of pages referenced in the last 4 minutes.

Table 5 presents the figures for the amount of read-sharing and write-sharing of pages referenced in the last second, and of those referenced in the last four minutes. The table presents the number of shared pages in each category, followed by a percentage. This is the percentage of the pages referenced in the time interval that were shared. Thus the table shows that 42 pages were referenced in the last second and were read-shared, and that 31% of all of the pages that were referenced in the last second were read-shared.

Table 5

Sharing of Pages

	Referenced in Last 1 Second	Referenced in last 4 Minutes
Read-Shared	42 (31%)	410 (24%)
Write-Shared	1 (1%)	10 (1%)

Notice that the pages that are more recently referenced are more likely to be shared. This is expected, as shared segments should be more frequently referenced.

We can also compute the average number of processes that share a page at each level of the memory hierarchy, just as we got an average for the number of processes that share a segment. This is done by computing the sum for each memory level of the number of processes that share each page at that level, and dividing that sum by the number of pages at that level to obtain an average. Table 6 presents these averages for pages of active segments and directories in core, on the paging device (PD), and on the disc.

Table 6

The Average Number of Processes Sharing a Page.

	Pages of Segments	Pages of Directories
Core Pages	7.6	27.6
PD Pages	4.0	16.8
Disc Pages	2.8	7.9

Notice that the number of processes that share a page is highest in the fastest portion of the memory hierarchy. This is because the chance that a page will be referenced increases with the number of processes that share that page, and the more heavily shared pages compete more effectively for space on the paging device and in core.

5) Conclusions.

The results above show that most of the sharing of memory pages is confined to read-sharing of system created segments. This fact suggests that it would at least be possible to implement a Multics-like system in a distributed environment where direct sharing of memory pages is not allowed. This could be done by providing each site in the distributed system with a copy of the system segments.

Writeable segments are in general shared intentionally only when synchronization is used to limit the writing to one process at a time. Such synchronization could be accomplished in a distributed system without direct sharing of memory pages.

The measurements also show that although the number of shared segments is small, a large proportion of the references to segments reference shared segments. There are several large segments that are shared among all of the processes on the M.I.T. Multics system. The figures show that the average page of main memory is shared among 11 processes, and the average bulk store page is shared among 5 processes. This means that as much as 11 times as much main memory and 5 times as much bulk store may be needed to achieve performance equal to the M.I.T. Multics without shared pages. These estimates are probably far too high, because the number of shared pages may be greatly overestimated. Following are some ideas for ways to get better measurements.

One idea that would be relatively easy to implement as an experiment would be not to turn on write access to a segment for a process until that process actually tries to write. This would allow us to obtain a much more accurate measurement of the number and nature of writeable segments, because it would provide a method to determine which processes have written an active segment. This change would cause each process that writes a segment to take

an extra fault in doing so, but would reduce the number of unencacheable segments. It is therefore uncertain what effect such a change would have on performance.

A much more difficult problem is that of trying to determine which pages a processes actually references. This probably cannot be done without some major changes, as all processes that share a segment share the same page table, leaving no place to record the use of pages of that segment by individual processes. It may, however, be possible to get some idea of how much a process uses the segments in its address space. This could be done either by making a hardware modification to turn on a bit in the sdw for a segment each time that that segment is used, or by periodically turning off access to segments in software, and recording a process's use of a segment when that process gets a fault trying to reference it. The hardware scheme has much less effect on performance, but would not be easy to experiment with. The software scheme is easy to experiment with, but is very inefficient.

I would greatly appreciate further suggestions for experiments. The appendix to this report describes the programs used to obtain most of the data, which are publicly available in the directory >udd>CSR>Montgomery>sharing.

APPENDIX A

Notes on the encacheability algorithm.

Each of the two processors on the M.I.T. Multics system has a private cache memory that holds the memory words most recently referenced by that processor. Because there are two such caches, segments that are shared and writeable must be prevented from entering either cache. All other segments can be allowed to enter either cache without any difficulties.

A perfect algorithm for deciding whether or not a segment should be allowed to enter a cache would mark as unencacheable exactly the set of shared, writeable segments. The algorithm used in Multics was therefore investigated as a possible source of data on write-sharing. This algorithm marked 6% of the active segments unencacheable in the snapshot from which the data for this report was taken.

6% is much larger than the reported percentage of write-shared segments. The reason for this difference is that the encacheability algorithm is not perfect. A segment is marked as unencacheable if it has been shared and writeable at any point since it was last activated. Thus the percentage of unencacheable segments is an approximation of the amount of write-sharing in the time window of the average active time for a segment. This time window is larger than that used in the other measurements. Thus many of the segments that were unencacheable were not even shared at the time of the snapshot.

Correcting this error in the encacheability algorithm may improve performance. Most of the data bases used by the Initializer are currently unencacheable, but are only rarely shared. Because the response of the Initializer is problem in the M.I.T. system, allowing the Initializer's data bases to enter the cache may improve performance.

APPENDIX B

Program Descriptions

Name: share

This command prints a number of meters of sharing in Multics derived from the AST and the AST trailer list.

Usage

share path {control args} {select args} {distribution args}

1) path is the pathname of the copy of the sst to be examined. Certain arguments require data in addition to the sst as noted below. The segments containing this data are searched for in the directory containing the segment specified by path. (Note that if path is a link, the directory containing the link is searched).

2) control args These arguments control the mode of operation of the command. If no control arguments are given, segments on the AST used lists will be selected according to the selection arguments and placed in categories according to the distribution arguments. Summaries of the selected categories are printed. The following control arguments are available and cause additional functions to be performed.

-names This argument causes the pathname of each selected segment to be printed. This argument requires that the data base sst_names_ be present in the directory that contains the sst being analyzed. If an empty segment is supplied for sst_names_, only the greater-than signs (>) in the pathname are printed.

-long This argument causes a brief summary to be printed for each selected segment. The summary consists of the uid, the number of pages of the segment in core, the number of pages on the PD, the total number of pages, a summary of the status flags for that segment, and a list of the processes which have trailers for the segment. The following status flags are printed:

U - The segment has been used since it was activated.
M - The segment has a page that has been recently modified.
W - The write_access_on flag.
C - The cache flag, (inhibits encaching segments involved in I/O).
A - The any_access_on flag.
D - The segment is a directory.

The process group ID of each process on the trailer list is printed where available. A unique index is used for each process that cannot be identified. This option needs the data base str_seg to print the trailer list. In addition, if process group IDs are to be produced, the segment tc_data is needed and as many as possible of

>scl>answer_table, >scl>aut, >scl>dut should be copied to the directory containing the sst being analyzed. Complaints are made about missing or inconsistent data bases.

-ratios This option causes the average number of processes sharing a segment, the average number of processes sharing a core page, the average number sharing a pd page, and the average number sharing a disc page to be printed for the selected group of segments.

-hist This option causes a histogram of the number of the selected segments shared by exactly n processes to be printed (n<51).

3) selection args These arguments select the segments to be used in the computation of sharing. The following selection arguments can be used:

-used,(-^used) Select segments that have been used since activation.

-mod,(-^mod) Select segments that have a page with ptw.phm on.

-tmod,(-^tmod) Select segments that have been modified within the last 5 minutes. (Note that the program records the time that it is first run on a particular copy of the AST, and figures the 5 minutes from then).

-write,(-^write) Select segments that are writeable by at least one process.

-any,(-^any) Select segments that are readable by one or more processes and not writeable, or readable and writeable by exactly one process.

-cache,(-^cache) Select segments that are inhibited from entering the cache because they have I/O in progress.

-core,(-^core) Select segments that have at least one page in core.

-share,(-^share) Select the segments initiated by 2 or more processes. (This option requires a copy of str_seg).

-dir,(-^dir) Select directories.

4) distribution args These arguments select the categories into which the selected segments will be sorted. If no distribution arguments are given, no summary of the selected segments is generated. For each pair of selection arguments described above, there is a corresponding distribution argument that does not have the leading -. Thus to see a breakdown of the active directories into categories of shared vs. unshared, one would give the arguments -dir and share.

Name: prune_sst

This command modifies a copy of the AST and AST trailer list to remove selected segments, directories, or processes.

Usage

prune_sst path -function- {select args}

1) path is the pathname of the copy of the AST to be pruned. The copy of the AST trailer segment is assumed to be in the same directory, with the name str_seg

2) function. This argument determines the mode of operation for the command. There are four modes of operation that are described below.

-init This mode causes the copy of the AST to be initialized for use with prune_sst and share. The circular links in the AST used list are broken, and inconsistent AST entries and AST trailers are removed. prune_sst should always be invoked with this mode prior to other pruning.

-delete_processes (-dp) This mode causes selected processes to be deleted from all trailers. The select args are taken as the names of the processes to be delete. All process names must be fully written out (as in Initializer.SysDaemon.z).

-delete_segments (-ds) This mode causes selected segments or directories to be deleted from the AST used lists. The select args are taken as the unique identifiers (uids) of the segments and directories to be deleted. Two other arguments can be used to control the selection of segments for deletion.

-tree This argument causes all segments and directories below the selected directories to be deleted as well.

-level n (-lv n) This argument specifies a level in the hierarchy below which all segments and directories will be deleted.

-select_segs (-ss) This mode causes all of the selected segments and directories to be retained, and all others to be deleted from the AST used lists. The arguments for selecting segments to be retained are the same as those used to select segments for deletion in -delete_segs mode.

Note:

This command is intended for use with the share command. It does not alter any fields of the AST entries and trailers used by that command but does destroy some of the other fields.

Name: restore_sst

This command undoes the pruning done by prune_sst.

Usage

restore_sst path

1) path is the pathname of the AST to be restored. The copy of the AST trailer segment is assumed to be in the same directory with the name str_seg.

Note:

This command does not undo all effects of prune_sst. It does not undo the effects of the -init mode of prune_sst, and it does not restore the fields of the AST entries damaged by prune_sst but not needed by prune_sst or share.