from M. [illegible]
12/11/70

[for MPM reference or ~~do~~ beginner's guide?]

Overview of Program Synthesis[*] ~~and Related Problems~~

Referencing and Finding Segments

In ~~its greatest generality,~~ As the term is used in Multics, program synthesis is the process operations of bringing together all the information necessary to enable a program to execute. This may mean only one procedure with internally contained data or it may mean bringing together several procedures which call upon one another and reference data contained in still other segments.

The problem dealt with here is how procedures reference each other and external data segments in an unambiguous way. The most direct answer to this problem in Multics would be ~~is~~ that all segment references be ~~are~~ by full path name, i.e., >udd>project>Doe>integrator. However, this answer leads to two ~~other~~ problems. First, if a segment be ~~were~~ moved from one directory to another then all procedures referencing that segment would need ~~have~~ to be recompiled, since they would contain the old ~~full~~ path name in their linkage section. Second~~ly~~, user ~~substitution of private~~ replacement with modules~~, e.g., for testing,~~ of his own in some collection of shared components becomes very difficult. These problems are further complicated by the fact that the sharing is both possible and encouraged in the Multics environment. For these reasons an allowed method of reference and the one normally coded into Multics procedures is through single component names (with no ">" symbol) called reference names. These are usually branch entry names.

What else could they be?

---

[*] See MPM Section I.2.4 and I.2.5

Examples of such names are calc, calc.pl1, MAM.calc.list.

~~The~~ this use of single component reference names does lead~~s~~ to a further difficulty. Single component names do not uniquely identify segments in the file system hierarchy. How then are reference names ~~used to reference~~ associated with path names of segments in the hierarchy? The remainder of this section ~~explained~~ explains the ~~consequences of~~ techniques of controlling this association.

Search Rules

When a reference name is used, a search is undertaken for a suitable item in the hierarchy to associate ~~[illegible]~~ with the reference name. This search is accomplished by ~~Reference names are used to search, the hierarchy by~~ prefixing successive directory pathnames to the reference name. ~~them~~ ~~during the search for the referenced segments.~~ In this way, the file system, as usual, is given ~~is allowed to use~~ full pathnames as potential identifiers of ~~to find~~ the segment~~s~~ in question.

To illustrate the ~~process~~ operation just described, suppose that user Doe has compiled two programs "A" and "B" and that their branches entries are in the directory >user_dir_dir>project>Doe. Further suppose that "A" calls "B". How is "B" located in the hierarchy at the time the call is made? Obviously, (to us) the "B" desired is ">user_dir_dir>project>Doe>B". Therefore, the system must somehow prefix ">user_dir_dir>project>Doe" onto the reference name "B".

The path name construction is accomplished by use of a previously ~~stated~~ provided list of ~~pathname prefixes.~~ directory names. The elements of this ordered list are called the search rules. ~~These prefixes are the path names of directories which are to be searched for the named entry.~~ The order of searching is the same as the order of the pathnames in the list. The searching terminates as soon as a match with an attempted pathname ~~the named entry~~ is found.

The system supplies a default set of search rules which is sufficient for most users; however, the sophisticated and knowledgeable user may provide a different ~~define a new~~ set of search rules to be used in his process. It is important to note that the most important system supplied commands lie in the directory whose pathname is >system_library_standard. This includes the command which changes the search rules. If the pathname ">system_library_standard" ~~is~~ is not included among the user's search rules and he does not have a copy of the command to reset the search rules, then the user may find himself in a situation in which he can neither change the search rules nor ~~or~~ request a new process nor ~~not~~ properly logout and start over. The user is, therefore, warned to be extremely careful in defining his own search rules. A typical reasonable set of search rules is: ~~could be:~~

1) [1*] search in the parent directory of the procedure making the reference.[2**]
2) [1*] Search the user's working directory.
3) ~~search the system libraries.~~ ~~[3***]~~
4) search the user's process directory.[4+]

---

meaning?

* These search rules are variable in that the caller and therefore its parent directory can change. Also, the working directory can be changed without changing the rules "search the working directory".

** This particular rule is dynamic since the parent may change with the caller during the execution of a single command.

re-orthogonalize

~~***~~ Search each of the system libraries starting with system_library_standard.

+ A per process temporary directory.

The following is an example of how the search rules might be used.

Suppose that user Doe is testing a new subroutine "A" which replaces subroutine "A" in a subsystem which belongs to user Smith and is in Smith's directory, >user_dir_dir>project>Smith. According to the default search rules given above when subroutine "A" is first referenced (by Smith's subsystem) the first directory searched will be the caller's parent directory, >user_dir_dir>project>Smith. If user Doe swaps rules 1) and 2) and then calls Smith's subsystem, the first directory searched when referencing subroutine "A" will be Doe's working directory thus allowing use of Doe's routine "A".

Of course, this is a little dangerous since any standard subsystem used by Doe will look first in Doe's working directory for routines referenced for the first time. If any routines in Doe's working directory have the same names as some standard system routines, incorrect references could occur together with much confusion.

The Scope of Names

When a segment is referenced and found, its name is entered into a table which we shall call the reference name table. In addition, the segment is assigned a number by which it can be referenced. When such a number has been assigned to a segment then the segment is said to be known ?

in the address space of the process. A segment in the address space of a process is always entered into the reference name table.

A segment in the address space of a process is said to be known to the process and entering a segment into the address space of a process is called making the segment known.

you defined this idea when one will do...

When a segment is referenced in Multics, the reference name table is searched first to see if the segment is already in the address space of the process or is known. This is done even before the search rules are applied. Thus, if some program references segment "X", the reference name table is searched for the name "X". It is clearly possible to have two segments "X" in two different directories. there are some cases in which It is also desirable to be able to reference both of these segments in the same process. For example, user A has a program which references his own segment "X" and then calls upon a program belonging to user B, ~~say, a statistical or an I/O package~~. If one of the modules of user B's program also has the name "X" then a difficulty arises since there is already a segment "X" in the address space of the process and "X" is already entered into the reference name table.

This conflict or duplication of names is diminished by limiting the scope of the names entered into the reference name table. The limit is imposed by associating a pathname with each name in the reference name table.

and requiring what?

The pathname associated with a reference name in this way is the pathname of the parent directory of the procedure making the reference. What mean? This means that a reference name is not recognized unless it is being referenced from a procedure in the same directory as that associated with the reference name in the reference name table. This looks suspiciously like the first search rule stated in the example above and is based on the same assumption: ~~Namely~~, ~~that~~ All the segments in a given directory are related at least to the extent ~~of not conflicting with each other. (This means~~ that if one module in a directory references a segment "X", no other module in that directory need to ~~will~~ ~~needs to~~ reference a different segment "X". Since the reference name table is searched before the search rules are applied, this amounts to a preemptive search rule. (A way around ~~this~~ preemption rule will be described below.)

There are three further comments to be made about known segment names. First, all directory names in the reference name table are the full true pathnames of the directories themselves. Second, when a command is typed on a console, the command processor program causes the command procedure to be made known, but it is working on behalf of the user. In addition, the user may sometimes wish to invoke more than one version of a command by the same name, perhaps for testing purposes. Therefore, when the command processor causes a command to be made known, the pathname associated with the reference name in the reference name table is the pathname of the parent directory of the command itself. Finally, a segment can have more than one name in the reference name table, either the same reference name with different associated pathnames or different reference names.

It sometimes occurs that a segment has been made known and the user wishes to make a new segment known in the same directory with the same name. In order to allow the new segment to be made known the previous segment with the same name must be made unknown, ("terminated") removed from the address space of the process. A set of commands and subroutines are available to allow the user this option.

Occasionally a user will wish to use a segment which cannot be found using his normal search rules. A command name ? and subroutine is available to allow the user to make a segment known using its full pathname and to associate with its reference name in the reference name table a pathname specified by the user. Thus, if user A wishes to test user B's subroutine with pathname ~~1/s~~ >udd>proj>B>sqrt (not found by user A's search rules) with a test program of his own with pathname >udd>proj>A>math>test_sqrt> then user A can ~~made~~ make the routine known and have its reference name associated with the appropriate directory. User A will make "sqrt" in >udd>proj>B known but the pathname associated with "sqrt" will be >udd>proj>A>math. When "test_sqrt" in directory >udd>proj>A>math calls "sqrt" the correct segment is then sure to be used.

An Example

Suppose that user A has a program "george" with pathname >udd>proj>A>george. Suppose that "george" calls "X" and that there is an "X" with pathname >udd>proj>A>X. When "george" calls "X" the reference name table is searched and no "X" is found. The search rules are invoked. The first search rule above says look in "george's" parent directory, >udd>proj>A. The segment "X" will be found and marked known. Associated with "X" in the reference name table will be the pathname >udd>proj>A because "george is in >udd>proj>A and "george" made the reference to "x".

Suppose that "george" now calls "Q" which is found using the search rules in >system library_standard. The segment >system_library_standard>Q will be made known and its reference name "Q" will be associated with the pathname >udd>proj>A, the pathname of the parent directory of "george".

Suppose that "george" now calls "Z" which is a link in >udd>proj>A to the segment >udd>proj>B>Tom and that "Tom" calls "X". The reference name table is searched and an "X" is found. But that "X" is associated with >udd>proj>A and so is not correct for this reference (remember >udd>proj>B>Tom is making this call to X). The search rules are invoked. Now suppose that the first segment "X" found is in >system_library_standard. Then >system_library_standard>X is made known and associated with >udd>proj>B. When execution returns to "george" then "george" calls "X" again. The reference name table is searched and an "X" is found associated with >udd>proj>A. This is the correct "X" for "george" since "george" is in >udd>proj>A and execution continues using the "X" so located.

Usage

The ability to set the search rules allows the user to replace some system routines, e.g., some math routines, with appropriate routines of his own. This would be done by putting the replacement routines in a special directory and adding a search rule to search that directory before searching the system libraries.

Details on how to set the search rules are given elsewhere but an outline will be given here.

If the user does nothing then a default set of search rules, ~~approxi-mately those given~~ outlined above will be used. ~~Sect~~ MPM section — details the directories searched.

If the user wishes to set his own search rules then he must create a segment. In this segment are listed the pathnames of segments to be searched in the order they are to be searched.

To invoke his search rules, the user calls the command with the argument thru pathname of his search rule segment containing directory pathnames. ?

The user can change his search rules by changing the pathnames or order of pathnames in this search rule segment and again calling the command to replace the old search rules with the new ones.

The user may at any time call a command to print the current operational search rules. This includes the time prior to setting any search rules himself, thus printing the default search rules.

<u>Design Notes</u> (Not for use with the above)

Needs an MPM section too

1) The syntax for the search rule segment will allow comments using /* */ separated by a blank from the pathname. The only special symbols allowed in pathnames are backspace ,! , . , ´, `, _, . , > and red and black shift.

2) Allowed are the following special symbols:

rdir	referencing directory
wdir	working directory
pdir	process directory
hdir	home directory
ldir	system libraries

3) Whenever the working directory is changed, the command processor memory will be erased.

4) It will be possible to have a different set of search rules for each ring.