

Debugging Tools

SYNOPSIS

This section describes at an introductory level the use of debugging tools that are available on most UNIX systems. Additional information on these tools for advanced use should be obtained from the *man* pages as well as hard copy manuals. **NOTE:** Alsys Ada and IBM Ada/6000 have their own debugging systems. Please refer to those sections when debugging Ada programs.

A WORD ABOUT COMPILING FOR DEBUGGING TOOLS

Before you can use a debugging tool on your program, you must compile your code with the appropriate compiler flag for the compiler you are using. In almost every case this will be the *-g* flag. The *-g* flag instructs the compiler to provide additional information within the object file it generates for the debugging tools.

In most cases, the debugging tool of choice is *dbx*.

A WORD OF CAUTION ABOUT DEBUGGING

When you compile a program with the debugging option selected, the additional code generated for the debugger can shift the portion of the code with the problem in memory. This movement of the troubled code can make the problem during a debugging session. If this occurs, you may be required to use more primitive techniques for debugging, such as adding statements that provide output of various variables that you suspect of causing the problem. This will be discussed later in this section under the title "MANUAL DEBUGGING TECHNIQUES".

A SIMPLE METHOD OF DEBUGGING

In most cases, a user only needs to find out where and why a program is aborting. This is probably the first level of debugging that should be attempted by novice users. To perform this task the user will need to perform several steps in the debugging process.

1. Compile your program(s) with the *-g* flag.
2. Execute the debugger as follows:

```
% dbx objectfile(s) <CR>
```

where *objectfile(s)* is one or more files that result from step 1.

3. At the *dbx*> prompt type:

```
dbx> run [arguments][>output  
redirection][<input redirection> <CR>
```

where "*arguments*" are arguments being passed to the program and "*>output redirection*" and "*<input redirection*" are for any required redirection for program execution.

4. Run the program as you normally would. When the error occurs, *dbx* will provide information regarding the location of the error, and possible reasons for its occurrence. It should also provide more detail about the error than the initial error messages you may have received without *dbx*.

THE DBX COMMAND

The **dbx** command has the following format:

```
% dbx [-c file][ -I dir][ -r][ -x dbx.name]
    [ -X in, out, error, pipe] objectfile [corefile] <CR>
```

In addition to the parameters show above, **dbx** also has a large number of sub-commands that may be issued at the **dbx>** prompt.

For further information on **dbx**'s parameters or sub-commands, please refer to the man page as follows:

```
% man dbx<CR>
```

MANUAL DEBUGGING TECHNIQUES

In some cases, the error may not be detectable with tools such as **dbx**. In this case, the user must use the tools of the language to provide the necessary information to pinpoint the problem. These tools fall into several categories.

A. TURN OFF ANY OPTIMIZATION

There are often times when optimization can adversely effect the code. This is primarily due to the style in which the code was written. Often, simply turning off any optimization will cure problems until they can be corrected.

B. INSERT OUTPUT STATEMENTS

If you are aware of your code and the logic behind it, it is often fairly easy to insert output statements within the code to determine such things as how far the execution has gotten and what are some of the variable values.

Be careful! Inserting statements can shift the problem around to the point where it may not be identifiable. Use such output statements sparingly.

DOCUMENT CODE: UNIX-60301A

DATE REVISED: August 25, 1994

NOTES