

Title:	Starting using GVD
Author:	Ian Johnson (August 2002)
Prerequisites:	A Linux based computer
Module:	UFS001C1
Awards:	B.Sc. CRTS / CSE / C&T
References:	Royce, T "C Programming", Macmillan, 1996 Kernighan, B & Ritchie, D, "The C Programming Language", Prentice-Hall 2 nd Ed. 1988 GVD Manual (online)

Introduction

For any serious programming a debugger is essential. The compiler will report syntax errors, but is totally unable to detect logical errors. As computers do what you say, rather than what you mean this can be a problem. At this point the debugger is your friend.

GVD is the Gnu Visual Debugger. It looks vaguely Microsoft like, and provides a GUI front end to gdb, the command line gnu debugger.

gdb is incredibly powerful, but is command-line driven and can be a bit of a pig to use. The Gnu Info system contains a gdb manual which you can peruse using the command `info gdb`.

A faulty program

In order to do some debugging we need a program with a logical error that will compile, but will run incorrectly. Hopefully, you will not be able to see the error in the following example, nor will the compiler find it.

Type in the program overleaf using emacs and save it to a file called `test1.c`

Compile the program using:

```
gcc -g -o test1 test1.c
```

The `-g` option tells gcc to generate debugging information.

test1.c

```

/*****
** PROGRAM: test1.c
** AUTHOR: Ian Johnson
** DATE: 24-Sept-2002
** COMMENTS: Deliberately buggy program
** To demonstrate GVD
*****/

#include <stdio.h>

main()
{
    char c, nl;

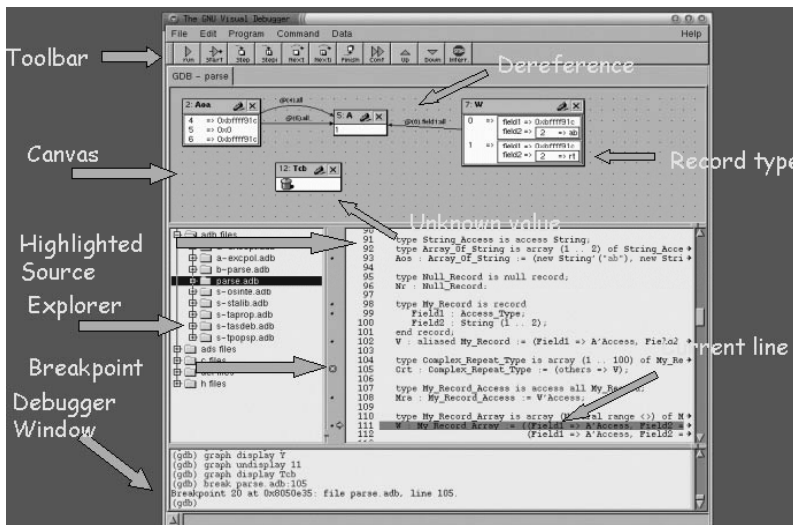
    c = getchar();
    nl = getchar();
    while (c != 'q')
    {
        if (c = 'a')
            printf("you entered a\n");
        else
            printf("you didn't enter a\n");
        c = getchar();
        nl = getchar();
    }
}

```

GVD

Gvd is started by simply typing `gvd`. It is probably a good idea to background it by typing `gvd &` though!

When you start `gvd`, you get the main window. The following diagram from the `gvd` manual explains what you see:



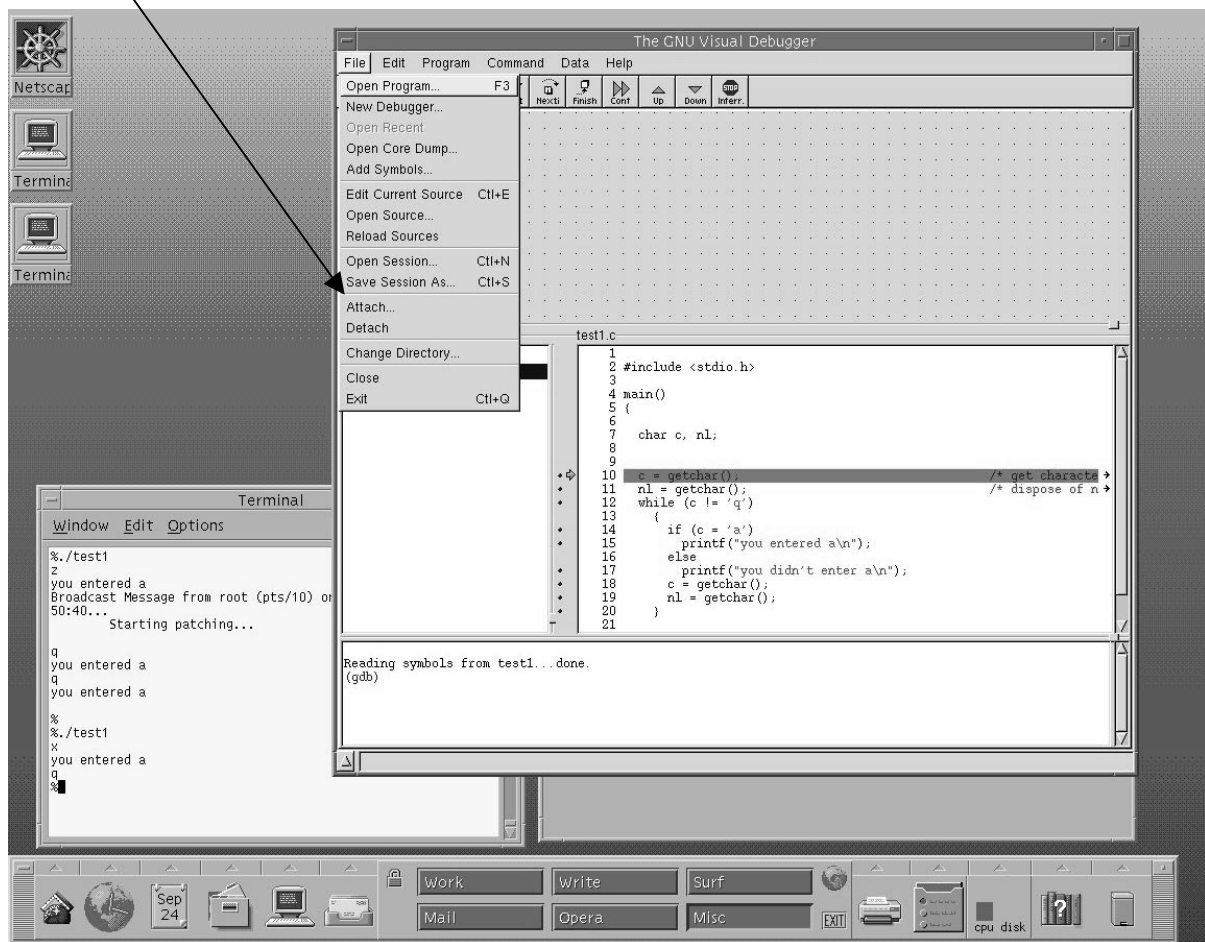
If you want to see the gvd manual and a clearer picture, click help then select gvd manual. It is well worth reading the friendly manual, since you will want to figure out how to do things on your own later.

For hardened gdb hackers, the debugger window allows you to directly type gdb commands in.

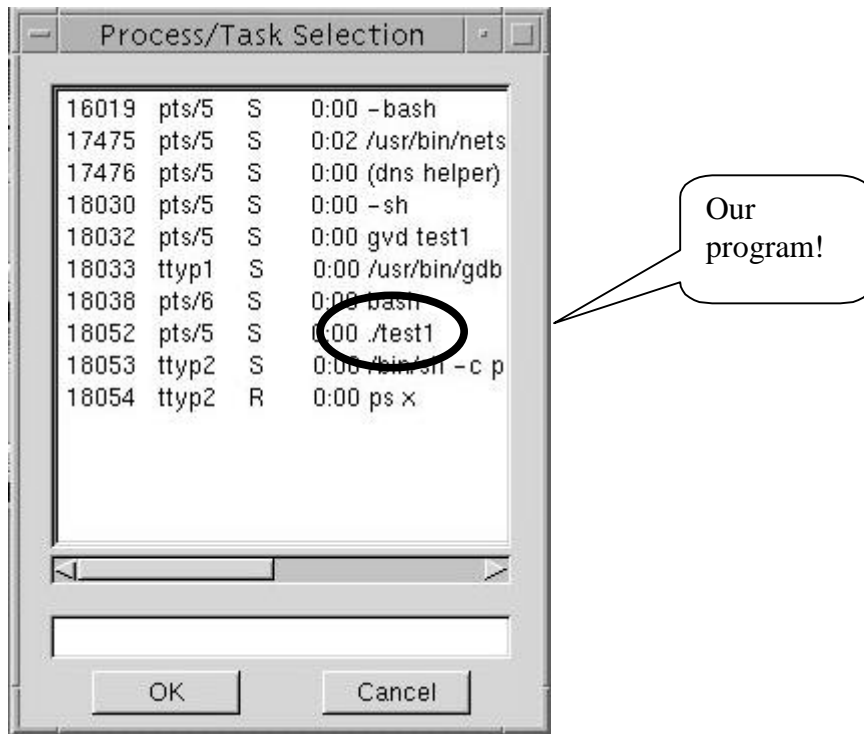
To debug our program, we will need to be able to run it whilst accepting input and seeing output. To do this we will need to run the program in a terminal window (e.g. xterm).

First, use File->Open Program to open test1 in GVD.

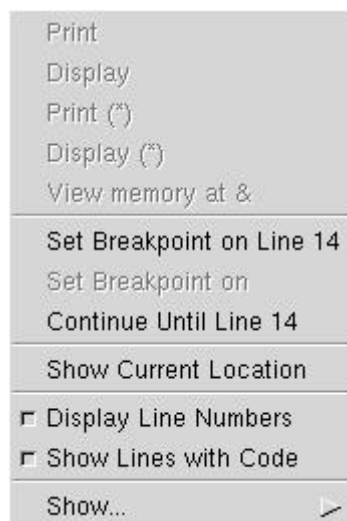
Now, start an xterm, and then in the xterm window type `./test1` to run the program. You have now created a process on your computer. To attach to this process click on file, then attach in GVD.



This will pop up the Process/Task selection window which contains output similar to that you would get from the unix ps command. (Try `man ps` for more information on ps). From this window, find your running program, select it by clicking on it, then click on OK. You'll notice that the process number is what is actually selected.

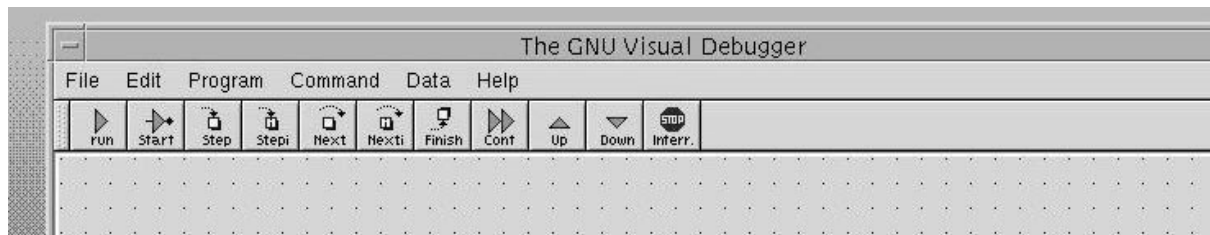


Once you have attached to the process you need to set a breakpoint where you want the program to pause execution once you have entered some data. Probably the best point is on the line with the while statement. Move your mouse over this line, right click and you should get a menu similar to this:



It shows the line number you've selected. If this is correct, click on **Set Breakpoint ...**. A red circle with a cross in it will appear next to the selected line. Now go to the top of the gvd window and click finish.

You can see the main buttons in the diagram overleaf:



Finish will run a process until you reach a breakpoint you've set or the end of your program.

Step steps a line at a time through your program, going into functions as necessary. If you enter a library function, you'll be pressing next a good few times to get back out. Next steps through without going into functions, while stepi & nexti do the same things but an instruction at a time rather than a line.

Now, go into your programs xterm, type a character say, x and press return. The gdb window will respond by moving the active line (pointed at by an arrow and highlighted green) to the line you selected to breakpoint at:

```

y
10  c = getchar();
11  nl = getchar();
12  while (c != 'q')
13  {

```

Now go to the Data pull down and select display local variables (or press Alt-L). A window will appear on the canvas similar to the one below:



This displays all of the local variables in the current function. We have two c, which has the decimal value 120. It also helpfully displays that this is the ASCII value of the character 'x'. The other variable is nl, which as intended holds the newline character we collected from the keyboard. Pressing the next button will now step you through your program a line at a time. If a variable changes value its colour in the info locals window will change to red. If you press next on an input line (in this program, the ones with getchar() calls) the wait cursor (clock face) will appear. Go to the window your program is running in enter a letter and press return, then go back to gdb, the wait cursor will have gone and you can press next again.

You should now be able to figure out what is wrong with this program. Explain the problem to your lab tutor, and then fix it. Now extend the program to handle several letters.