Using ddd with postgres on the instructional computers

CS186 Staff
University of California at Berkeley
September 6, 2005

1 Assumptions

For the purpose of this document I will assume that we are working with the postgres source tree that has been setup for Project 1. Please follow the instructions in Section 3 of the Project 1 document carefully. In Project 1 you will have learn how to start the postmaster with pg_ctl and create a database with createdb. This document will assume the existence of a database called test. Using the test database, we perform the actions listed below to create a table T, populate it with data (in this case we load it with 100000 rows) and then stop the postmaster with pg_ctl stop. (These steps take time!).

pentagon> cd
pentagon> pg_ctl start
postmaster successfully started
pentagon> psql -f ~cs186/gendata/sql/tddl.sql test
psql:/home/ff/cs186/gendata/sql/tddl.sql:1: ERROR: table "t" does not exist
CREATE
COPY
pentagon> pg_ctl stop
waiting for postmaster to shut down...DEBUG: smart shutdown request
....done
postmaster successfully shut down
pentagon>

2 Using ddd

There are two ways to debug postgres:

1. Interactive mode
2. Bare backend mode

The interactive mode is where you start up the postmaster process (with pg_ctl), use psql to connect to it, which spawns off a separate postgres process that is dedicated to serving requests from the psql client. This is the mode that the scripts in the Hw1/MyStuff/PerformanceEvaluationScripts directory use, and the mode you used in Hw0. In the second mode, you don’t create a separate postmaster process and don’t use psql at all. Instead, you start up postgres from the command line and directly interact with it. While the latter is a very terse way of using postgres it has the advantage of being very debugger (like ddd) friendly.

This document will restrict itself to showing how to use ddd with postgres in the bare backend mode. If you really really want to use ddd with postgres in the server mode please talk to your TA for more help. This is not intended to be a replacement for ddd documentation. Think of it instead as a sample to guide you.

1. First check to make sure ddd is in your path.

pentagon> which ddd
/usr/swa/bin/ddd
pentagon>
If you can’t find ddd contact the course staff or your colleagues for help.

2. Startup ddd with the newly compiled and installed version of postgres.

```
pentagon> ddd bin/postgres &
```

The ddd “splash screen” would have come up (albeit a little slowly) and a few dialog boxes will open up. Read them and click anything you find interesting if you want to explore. Otherwise click on “close” until the main ddd window comes up. This should look something like Figure 1.

![Figure 1: Initial ddd screen](image)

I prefer having the toolbar embedded in the source window. So I click on Edit->Preferences and modify the dialog box as in Figure 2. There are other settings that you might want to change as well. Remember to use Edit->Save Options to save your settings!

![Figure 2: ddd preferences dialog box](image)

3. Now you are all set to start examining functions. We will start by looking at the function `GetFreeBuffer`. Clear what’s in the topmost pane (it probably has `main` in it) and type `GetFreeBuffer` instead. Then click on the Lookup button immediately to the right of this pane. This should look something like Figure 3.
4. At last we are all set to actually run postgres! You can do this either by hitting F2 or by using the menu option Program->Run. This will first result in a dialog box where you should type test. This will look like Figure 4. Click on Run and your ddd screen will now look something like Figure 5. Note the backend prompt in the third pane. This is the postgres prompt in the bare backend mode. This is different from the prompt you are familiar with when using psql.

5. Next we place a breakpoint in the function GetFreeBuffer by clicking (middle pane) on the first line of code in the function after the variable declarations. You can either right click and choose the set breakpoint or click on the line and then click on the Break button (a little to the right of the Lookup button). Your ddd should now look something like Figure 6. You might have to hit enter to see the backend prompt again.

6. Now click to the right of the backend prompt and enter select count(*) from T;,. This will cause the backend to run your query and stop at the breakpoint. In the middle pane you will see a green arrow pointing at the “stop” sign (the breakpoint). Now assume we’ve set a breakpoint at the beginning of the function StrategyBufferLookup, which is located in freelist.c. When the execution thread reaches this function, a trap will cause ddd to stop and your ddd screen will look something like Figure 7.

7. At this point you are ready to rumble. You might want to try one or more of:
   - Look at the current state of your stack (Use Status->Backtrace) - something like Figure 8)
   - Display the local variables (Use Data->Display Local Variables or Alt-L) and display the arguments (Use Data->Display Arguments or Alt-U). The result should be something like Figure 9 after you step through a few instructions.
• Display any expression you see fit. Just use your mouse to highlight any expression on the code. You will see the expression show up in the top pane. Either click on the Display button (to the right of the Break button) or right click on the highlighted expression and choose the appropriate option.

That’s it! You’re all set to go ahead. Play with ddd - a debugger is useful for more than just finding errors. It’s also extremely useful in stepping through code that you might not be changing just to give you context. The displays in ddd are extremely useful for understanding data structures. Any time you see a pointer you can typically double click on it and see the structure it points too. If it’s a void pointer and you know the actual type, you can type cast it by right-clicking on the pointer in the data pane and choosing New Display->Other.

Good luck and let us know if you have problems.
Figure 6: ddd after placing a breakpoint in GetFreeBuffer
Figure 7: ddd after breaking in StrategyBufferLookup

Figure 8: Examine the function call stack in ddd
Figure 9: ddd displaying local variables and arguments