Lab 7 : CS 387 Jan-Apr 2024

(1) Compilation of postgres server from source code, (2) Stepping through code in a debugger, (3) A simple modification to postgres source code

Overview

Postgres is an open source database server. In this assignment you will compile postgres from its source, install and run it. You will then execute simple queries on your compiled server, while stepping through the code in a debugger (VSCode). Finally, you will make a simple modification to the postgres source code to get a feel for the process of changing, compiling, and running the server.

Part-A Docker environment setup

1. Postgres source code is open source and can be downloaded from the official git repository – https://git.postgresql.org/git/postgresql.git or https://github.com/postgres/postgres.git
2. To compile postgres, you will need the right environment and a set of libraries
3. I have provided you with the needed Dockerfile which will setup the correct environment and libraries for you
4. Downloading the postgres source from its git repository can take some time, so I have also done the git cloning and provided you with a .tgz of the cloned directories/files
5. To setup the compilation and postgres running environments, you will also need some bash environment variables – I have provided these in env.sh – you can notice that the env.sh is run as part of your .bashrc, which in turn is setup by the Dockerfile while running runonceinit.sh
6. The steps for the environment setup are as follows
7. Download the given Dockerfile, postgres.tgz, env.sh, and runonceinit.sh – all into the same directory
8. Then run: **`docker build -t pgdev:v1 -f Dockerfile .`** (don't miss the dot at the end)
9. To run the image: **`docker run --name pgdevcontainer -it pgdev:v1`**
10. To start the docker container: **`docker start pgdevcontainer`**

11. To get into a bash shell of the running container: **`docker exec -it pgdevcontainer bash`**
12. Within the bash shell, you can check that there is postgres.tgz . You can also check the environment variables setup by env.sh – e.g. **`echo $POSTGRES_INSTALLDIR`** to check where postgres will be installed (in the next step, after compilation)

Part-B Compilation of postgres

1. Get into a bash shell of the running container: **`docker exec -it pgdevcontainer bash`**
2. Configure PostgreSQL for debugging as follows
3. Untar postgres and enter directory: **`tar -xzvf postgresql-16.2.tar.gz && mv postgresql-16.2 postgres && cd postgres/src`**
4. First edit (using vim) the file configure and replace -O2 in CFlags by -O0 in all occurrences except BITCODE_CFLAGS (there are 6 places)
5. Then run the following:

   `./configure --prefix=${POSTGRES_INSTALLDIR} --enable-debug`

   `export enable_debug=yes`

   NOTE: If you change the configuration (e.g. O2 above) after running configure, do a

   `make distclean`

   and then run configure again as above
6. Compile the PostgreSQL code as follows

   `make | tee gmake.out`
7. Install postgres:

   `make install | tee gmake_install.out`
8. Observe that the binary files are installed in the directory indicated by **`echo $POSTGRES_INSTALLDIR`**
9. Also observe the values of the environment variables **`$PATH`** , **`$LD_LIBRARY_PATH`** , and **`$PGDATA`**

10. Create a new PostgreSQL database cluster by using initdb as below. The command initdb below is in the install/bin directory, which should already be in your PATH

    `initdb -D ${PGDATA}`

11. Note: A database cluster is a collection of databases that are managed by a single server instance.

12. Run the postgres server: The command pg_ctl below is in the install/bin diretory

    `pg_ctl -D $PGDATA -l logfile start`

    Alternatively, you can also use:

    `postmaster -D $PGDATA > logfile 2>&1 &`

13. Create database using: **`createdb test`**

14. Run psql using: **`psql test`**

15. You can now type SQL commands. You can also type \? to see what are the available other commands.

16. You can stop the postgres server using: **`pg_ctl stop`**

17. To run postgresql in single user mode (very useful for debugging when you modify the sources), run it as

    `postgres --single -D ${PGDATA} test`

    Here test is the database you created

    Now you can directly enter SQL commands, instead of connecting via psql.

18. You can now run sql commands. E.g.

    ```
    create table foo(i int);

    insert into foo values (1);

    select * from foo;
    ```

19. You are now ready to explore the postgres source code further

Part-C Code step-through using a debugger

1. Note that postgres code is in a container, but the container does not have any sophisticated IDE like VSCode – but we can setup VSCode on your host machine, to

access the postgres source code inside the container

2. To access code in container in VSCode, install the "Dev Containers" VSCode extension package

3. Then click on >< button in bottom left corner, and select "attach to running container" and then select pgdevcontainer

4. Now you can add the folder “postgres/src” to your project – you should also save the project under a certain name

5. If you are not already familiar, you should familiarize yourself with actions like finding a symbol, finding definition of a symbol, finding usages of a symbol, etc. using VSCode

6. As a practice exercise, find the functions PostgresMain, exec_simple_query, pg_plan_queries – you must use the VSCode menu options to do this, instead of manually searching through the huge set of files – that is one of the things an IDE is for – easily browsing through a large code-base

7. Next you can setup the debug environment using a run configuration. Select menu option Run --> Start Debugging

    a. It will open prompt to create launch.json

    b. Click "Add Configuration" and select "C++ (launch)"

    c. Set "program": "/home/pgdev/postgres/install/bin/postgres",

    d. Set "cwd": "/home/pgdev/postgres",

    e. Set "args": ["--single", "-D", "/home/pgdev/postgres/install/data", "test"],

    f. Everything else is default

    g. You have now setup the debug environment while postgres is running in single user mode

8. Now select menu option Run --> Start Debugging again

9. You will get the SQL prompt corresponding to the postgres server running in single user mode

10. You should be able to step-through the postgres code as it runs

11. Set some break-points, e.g. at exec_simple_query, pg_plan_queries . You can run the SQL query **`select * from foo;`** and step through the code as it executes in postgres

12. Isn’t it amazing that you can touch and feel one of the most popular and successful database servers so easily? I think so :-) Very inspiring!

13. Let us get more inspired by making a simple modification to postgres

Part-D A simple modification to postgres source code

1. This is just a toy exercise to get a feel for the change-compile-test-execute cycle of a large open-source code-base, in this case, the postgres database server
2. First create two tables

   `r(a varchar(10) primary key, b varchar(10))`

   and

   `delta_r(op char(1), a varchar(10), b varchar(10))`
3. Your goal is to ensure that whenever a tuple is inserted into table r, the same tuple is inserted into delta_r, with op having value 'I'
4. In the postgres source code change, you only need to handle query strings:

   `insert into r values (...)`

   Anything that does not start with the exact string "**`insert into r values`**" can be left to the default processing of the existing code
5. While exploring postgres using the debugger, set a breakpoint at the function exec_simple_query and find the contents of query string.
6. Next, if the query string is an insert into table r, modify it by appending the extra insert into delta_r. Note that in postgres, a query string can have multiple commands separated by semicolons.
7. Ensure that your changed postgres server works for the insert queries as expected above. Also ensure that other queries have the default behaviour.
8. Submit ONLY the changed C file (exactly one file).