How to Write PostgreSQL Extensions

PostgreSQL Development Bootcamp
March 19, 2023
Anastasia Lubennikova, PostgreSQL Major Contributor, Neon, Inc.

What's on our list for today?

- Setup development environment
- Write an extension with C functions
- Advanced extension development topics

join master-class on `https://collabedit.com`

Setup the environment (Git)

Working with Git

```
git clone git://git.postgresql.org/git/postgresql.git
cd postgresql

git branch my_branch
git checkout my_branch

...

git add .
git commit

git format-patch -v1 origin/master
```

Setup the environment (configure)

Configure options

```
export WORKSPACE_PATH=/home/anastasia
```

```
./configure --enable-cassert --enable-debug --enable-depend
CFLAGS="-ggdb -Og -g3 -fno-omit-frame-pointer"
--prefix=$WORKSPACE_PATH/postgres_bin
```

Don't modify files generated by configure!

```
src/include/pg_config.h
src/include/pg_config.h.in
```

Set environment variables

```
PATH=$WORKSPACE_PATH/postgres_bin/bin:$PATH
```

```
LD_LIBRARY_PATH=$WORKSPACE_PATH/postgres_bin/lib
```

```
PGDATA=$WORKSPACE_PATH/postgres_data
```

Build postgres

```
    make -j4 -s && make install -s
or
    make world -s    // To build also docs and contribs
    make install-world -s
```

Before rebuilding:

```
    make clean       // Preserves ./configure results
    make distclean   // Cleans ./configure results
```

Create a cluster

pg_ctl

```
    pg_ctl initdb -D $PGDATA

    pg_ctl -D $PGDATA -l logfile start


    psql postgres


    pg_ctl -D $PGDATA stop
```

Run regression tests

Running the tests

```
    make check
or
    make installcheck    // Use pre-existing cluster
or
    make check-world     // To run all test suites


less src/test/regress/regression.diffs
```

Scripts for steps above

- https://github.com/afiskon/pgscripts

Debugging with GDB

```
postgres=# select pg_backend_pid();
 pg_backend_pid
----------------
        2512670
```

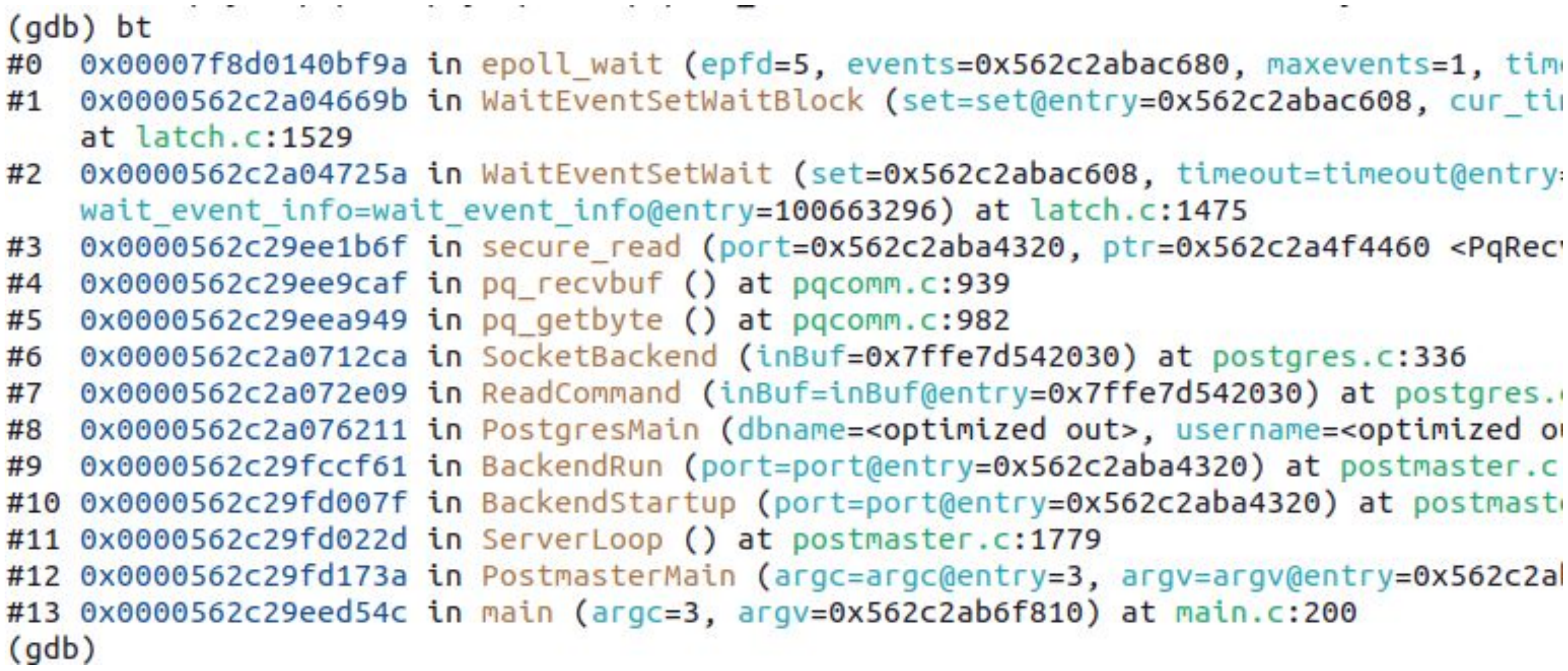
Connect with gdb:

```
gdb -p 2512670
    break ProcessUtility
    break ProcessQuery
```

Debugging with GDB

https://wiki.postgresql.org/wiki/Developer_FAQ#gdb

Getting a stack trace of a running PostgreSQL backend on Linux/BSD



```
(gdb) bt
#0  0x00007f8d0140bf9a in epoll_wait (epfd=5, events=0x562c2abac680, maxevents=1, tim
#1  0x0000562c2a04669b in WaitEventSetWaitBlock (set=set@entry=0x562c2abac608, cur_ti
    at latch.c:1529
#2  0x0000562c2a04725a in WaitEventSetWait (set=0x562c2abac608, timeout=timeout@entry
    wait_event_info=wait_event_info@entry=100663296) at latch.c:1475
#3  0x0000562c29ee1b6f in secure_read (port=0x562c2aba4320, ptr=0x562c2a4f4460 <PqRec
#4  0x0000562c29ee9caf in pq_recvbuf () at pqcomm.c:939
#5  0x0000562c29eea949 in pq_getbyte () at pqcomm.c:982
#6  0x0000562c2a0712ca in SocketBackend (inBuf=0x7ffe7d542030) at postgres.c:336
#7  0x0000562c2a072e09 in ReadCommand (inBuf=inBuf@entry=0x7ffe7d542030) at postgres.
#8  0x0000562c2a076211 in PostgresMain (dbname=<optimized out>, username=<optimized o
#9  0x0000562c29fccf61 in BackendRun (port=port@entry=0x562c2aba4320) at postmaster.c
#10 0x0000562c29fd007f in BackendStartup (port=port@entry=0x562c2aba4320) at postmast
#11 0x0000562c29fd022d in ServerLoop () at postmaster.c:1779
#12 0x0000562c29fd173a in PostmasterMain (argc=argc@entry=3, argv=argv@entry=0x562c2a
#13 0x0000562c29eed54c in main (argc=3, argv=0x562c2ab6f810) at main.c:200
(gdb)
```

Enable core dumps

Core Dumps — How to enable them?

Check & change core dump size limit:

```
    ulimit -a

    ulimit -S -c unlimited
```

Check & change core_pattern:

```
    sysctl kernel.core_pattern

    sudo sysctl -w
kernel.core_pattern=/coredumps/core-%e-%s-%u-%g-%p-%t
```

Extension

- PGXS - Extension Building Infrastructure
- PGXN https://pgxn.org/ - PostgreSQL extension network
- Built-in extensions `src/contrib/`

```
postgres=# select * from pg_extension ;
-[ RECORD 1 ]--+------------
oid            | 12796
extname        | plpgsql
extowner       | 10
extnamespace   | 11
extrelocatable | f
extversion     | 1.0
extconfig      |
extcondition   |
```

Manage extension

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]

ALTER EXTENSION name UPDATE [ TO new_version ]
ALTER EXTENSION name SET SCHEMA new_schema
ALTER EXTENSION name ADD member_object
ALTER EXTENSION name DROP member_object
```

DROP EXTENSION [IF EXISTS] name [, ...] [CASCADE | RESTRICT]

Kinds of extensions

- Package of SQL objects
  - Simple installation and version management
  - Extension dependencies
- Extension with C code
  - same as above +
- Extensions can reserve shared memory or LWLocks.
  Such extensions must be preloaded via shared_preload_libraries.

  -
    ```
    shared_preload_libraries = 'pg_stat_statements,
    ```

More cool stuff

SO YOU WANT TO MAKE AN EXTENSION? by Keith Fiske

http://slides.keithf4.com/extension_dev/#/20

Extension files

Packaging Related Objects into an Extension

pg_example.control – control file

pg_example.c – source code

Makefile

pg_example--1.0.sql – SQL script to create objects

sql/pg_example.sql – regression test

expected/pg_example.out – expected test results

README.md

Extension with C code

C-Language Functions

```
PG_MODULE_MAGIC;
```

```
C-function calling convention
```

PG_INIT

```
void _PG_init(void)
{
   /* Define custom GUC variables */
   /* Install hooks */
}
```

Custom GUC variable

```
#include "utils/guc.h"

DefineCustomBoolVariable("pg_stat_statements.track_utility",
      "Selects whether utility commands are tracked by
   pg_stat_statements.",
                           NULL,
                           &pgss_track_utility,
                           true,
                           PGC_SUSET,
                           0,
                           NULL,
                           NULL,
                           NULL);


MarkGUCPrefixReserved("pg_stat_statements");
```

Hooks

- Unofficial documenation https://github.com/taminomara/psql-hooks
- `Getting on a hook or PostgreSQL extensibility (2021)`
- `i.e. contrib/pg_stat_statements`

```
prev_ProcessUtility = ProcessUtility_hook;

ProcessUtility_hook = pgss_ProcessUtility;
```

Superuser-only

```
if (!superuser())
    ereport(ERROR,
            (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
             errmsg("must be superuser to run this
function")));
```

Check PostgreSQL version

```
#if PG_VERSION_NUM >= 150000
#include "miscadmin.h"
#endif
```

SPI

https://www.postgresql.org/docs/devel/spi.html

- Server Programming Interface that allows you to call SQL from C code.
- Used in procedural languages (PL/pgSQL, PL/python, PL/perl, PL/R ...)
- Some examples are in `contrib/spi`

```
#include “executor/spi.h”
```

SRF

[Returning sets](#)
`src/include/funcapi.h`

- Returns an iterator
- Requires careful memory handling to avoid memory leak

```
loop {
if (SRF_IS_FIRSTCALL)
    SRF_FIRSTCALL_INIT

SRF_RETURN_NEXT
}
SRF_RETURN_DONE
```

- `i.e. generate_series()`
- `contrib/pageinspect functions`

Background worker

https://www.postgresql.org/docs/devel/bgworker.html

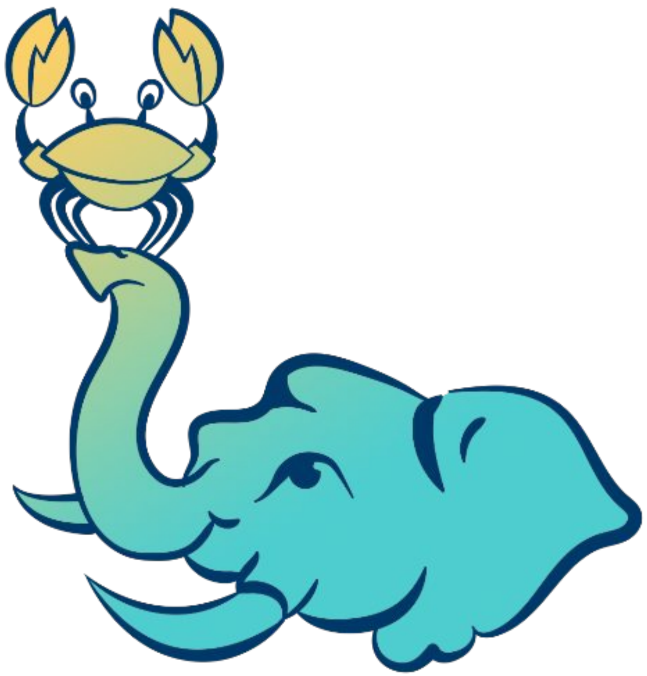
```
#include "postmaster/bgworker.h"
```

i.e. https://github.com/citusdata/pg_cron

pgx

pgx is a framework for developing PostgreSQL extensions in Rust

https://github.com/tcdi/pgx



Thank you for attention!
Questions?