Serverside Programming in C

Heikki Linnakangas / Pivotal

Stuff you can do in C

Extensions:

- SQL functions
  - or triggers, event triggers, operators
- Foreign Data Wrappers
- Hooks
  - Executor hook, planner hook, etc.
- Background worker processes

Stuff you can do in C

Hacking on PostgreSQL itself

- It’s open source. Post a patch. Or not. Up to you.

Getting started

- Get the PostgreSQL source code
  - git clone git://git.postgresql.org/git/postgresql.git
- Text editor
- Copy-paste an existing extension for the boilerplate
  - something simple from github
  - or from contrib/

```c
/*-------------------------------------------------------------------------
 *
 * colour.c
 *     A demo data type for colours
 *
 * Portions Copyright (c) 1996-2013, PostgreSQL Global Development Group
 *-------------------------------------------------------------------------
 */

#include "postgres.h"

#include "fmgr.h"
#include "utils/builtins.h"

PG_MODULE_MAGIC;

Datum colour_out(PG_FUNCTION_ARGS);
Datum colour_in(PG_FUNCTION_ARGS);

Datum red(PG_FUNCTION_ARGS);
Datum green(PG_FUNCTION_ARGS);
Datum blue(PG_FUNCTION_ARGS);

PG_FUNCTION_INFO_V1(colour_out);
PG_FUNCTION_INFO_V1(colour_in);
PG_FUNCTION_INFO_V1(red);
PG_FUNCTION_INFO_V1(green);
PG_FUNCTION_INFO_V1(blue);

Datum
colour_out(PG_FUNCTION_ARGS)
{
    Int32      val = PG_GETARG_INT32(0);
    char      *result = palloc(8);

    snprintf(result, 8, "#%06X", val);
    PG_RETURN_CSTRING(result);
}

Datum
colour_in(PG_FUNCTION_ARGS)
{
    const char *str = PG_GETARG_CSTRING(0);
    int32       result;
```

Headers

```
/*-------------------------------------------------------------------
 *
 * colour.c
 *    A demo data type for colours
 *
 * Portions Copyright (c) 1996-2013, PostgreSQL Global Development Group
 *-------------------------------------------------------------------
 */

#include "postgres.h"

#include <stdio.h>

#include "fmgr.h"
#include "utils/builtins.h"

...
```

Headers

```
/*-------------------------------------------------------------------------
 *
 * colour.c
 *    A demo data type for colours
 *
 * Portions Copyright (c) 1996-2013, PostgreSQL Global Development Group
 *-------------------------                                    -----------
 */

#include "postgres.h"

#include <stdio.h>

#include "fmgr.h"
#include "utils/builtins.h"

...
```

1. postgres.h

2. system headers, if any

3. PostgreSQL server headers (in alphabetical order)

C function

```c
/*
 * box_circle()
 *
 * Convert a box to a circle.
 */
Datum
box_circle(PG_FUNCTION_ARGS)
{
    BOX         *box = PG_GETARG_BOX_P(0);
    CIRCLE      *circle;

    circle = (CIRCLE *) palloc(sizeof(CIRCLE));

    circle->center.x = float8_div(float8_pl(box->high.x, box->low.x), 2.0);
    circle->center.y = float8_div(float8_pl(box->high.y, box->low.y), 2.0);

    circle->radius = point_dt(&circle->center, &box->high);

    PG_RETURN_CIRCLE_P(circle);
}
```

C function

Datum is used to pass around "column values"

```
/*
 * box_circle()
 *
 * Convert a
 */
Datum
box_circle(PG_FUNCTION_ARGS)
{
    BOX         *box = PG_GETARG_BOX_P(0);
    CIRCLE      *circle;

    circle = (CIRCLE *) palloc(sizeof(CIRCLE));

    circle->center.x = float8_div(float8_pl(box->high.x, box->low.x), 2.0);
    circle->center.y = float8_div(float8_pl(box->high.y, box->low.y), 2.0);

    circle->radius = point_dt(&circle->center, &box->high);

    PG_RETURN_CIRCLE_P(circle);
}
```

C function

```
/*
 * box_circle()
 *
 * Convert a box to a circle.
 */
Datum
box_circle(PG_FUNCTION_ARGS)
{
    BOX         *box = PG_GETARG_BOX_P(0);
    CIRCLE      *circle;

    circle = (CIRCLE *) palloc(sizeof(CIRCLE));

    circle->center.x = float8_div(float8_pl(box->high.x, box->low.x), 2.0);
    circle->center.y = float8_div(float8_pl(box->high.y, box->low.y), 2.0);

    circle->radius = point_dt(&circle->center, &box->high);

    PG_RETURN_CIRCLE_P(circle);
}
```

Arguments are passed in a special Function API style

and returned like this

Exposing C function in SQL

- CREATE FUNCTION … AS … LANGUAGE C

Or better yet...

- Package as an extension
- Examples in contrib/

Let’s get started

Ok, we’re down to C now.

What can we do?

- Single-threaded
- Code runs in a backend process

What **not** to do

- malloc/free (use MemoryContexts)
- Open files (use BasicOpenFile() etc)
- Signal handlers
- Fork (background workers)
- Launch threads (background workers)
- C++ exceptions (ereport())

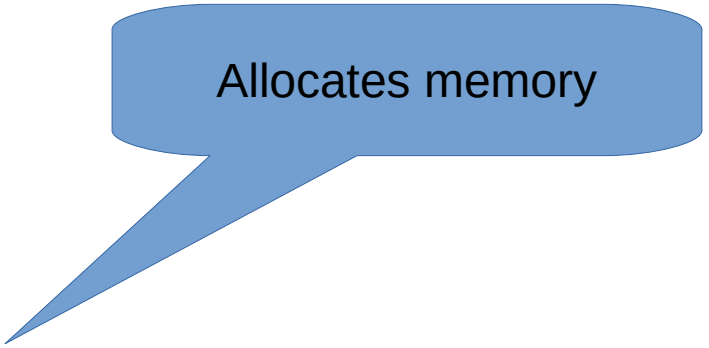
Memory Contexts

Memory Contexts

- Known as “memory pools” in other systems
- Tree hierarchy
- No need to free every little thing
  - They will be freed automatically when the memory context is destroyed
- CurrentMemoryContext global variable points to the currently active one.
- MemoryContextSwitchTo() to change context

palloc()

```
/*
 * box_circle()
 *
 * Convert a box to a circle.
 */
Datum
box_circle(PG_FUNCTION_ARGS)
{
    BOX         *box = PG_GETARG_BOX_P(0);
    CIRCLE      *circle;

    circle = (CIRCLE *) palloc(sizeof(CIRCLE));

    circle->center.x = float8_div(float8_pl(box->high.x, box->low.x), 2.0);
    circle->center.y = float8_div(float8_pl(box->high.y, box->low.y), 2.0);

    circle->radius = point_dt(&circle->center, &box->high);

    PG_RETURN_CIRCLE_P(circle);
}
```



MemoryContextStat()

```
TopMemoryContext: 79496 total in 6 blocks; 14400 free (8 chunks); 65096 used
  TopTransactionContext: 8192 total in 1 blocks; 7744 free (0 chunks); 448 used
  RowDescriptionContext: 8192 total in 1 blocks; 6896 free (0 chunks); 1296 used
  MessageContext: 16384 total in 2 blocks; 6648 free (1 chunks); 9736 used
  Operator class cache: 8192 total in 1 blocks; 560 free (0 chunks); 7632 used
  smgr relation table: 16384 total in 2 blocks; 4600 free (2 chunks); 11784 used
  TransactionAbortContext: 32768 total in 1 blocks; 32512 free (0 chunks); 256 used
  Portal hash: 8192 total in 1 blocks; 560 free (0 chunks); 7632 used
  TopPortalContext: 8192 total in 1 blocks; 7664 free (0 chunks); 528 used
    PortalContext: 1024 total in 1 blocks; 600 free (0 chunks); 424 used
      ExecutorState: 8192 total in 1 blocks; 3024 free (0 chunks); 5168 used
        printtup: 8192 total in 1 blocks; 7936 free (0 chunks); 256 used
        ExprContext: 8192 total in 1 blocks; 7664 free (0 chunks); 528 used
  Relcache by OID: 16384 total in 2 blocks; 4552 free (2 chunks); 11832 used
  CacheMemoryContext: 524288 total in 7 blocks; 11656 free (1 chunks); 512632 used
    index info: 2048 total in 2 blocks; 728 free (2 chunks); 1320 used: pg_db_role_setting_databaseid_rol_index
    index info: 2048 total in 2 blocks; 728 free (2 chunks); 1320 used: pg_user_mapping_user_server_index
    index info: 1024 total in 1 blocks; 48 free (0 chunks); 976 used: pg_user_mapping_oid_index
...
    index info: 1024 total in 1 blocks; 48 free (0 chunks); 976 used: pg_ts_parser_oid_index
    index info: 2048 total in 2 blocks; 680 free (1 chunks); 1368 used: pg_attribute_relid_attnum_index
    index info: 2048 total in 2 blocks; 952 free (1 chunks); 1096 used: pg_class_oid_index
  WAL record construction: 49768 total in 2 blocks; 6368 free (0 chunks); 43400 used
  PrivateRefCount: 8192 total in 1 blocks; 2624 free (0 chunks); 5568 used
  MdSmgr: 8192 total in 1 blocks; 7336 free (0 chunks); 856 used
  LOCALLOCK hash: 8192 total in 1 blocks; 560 free (0 chunks); 7632 used
  Timezones: 104120 total in 2 blocks; 2624 free (0 chunks); 101496 used
  ErrorContext: 8192 total in 1 blocks; 7936 free (0 chunks); 256 used
Grand total: 1066920 bytes in 158 blocks; 180008 free (75 chunks); 886912 used
```

```
TopMemoryContext: 79496 total in 6 blocks; 14400 free (8 chunks); 65096 used
  TopTransactionContext: 8192 total in 1 blocks; 7744 free (0 chunks); 448 used
  RowDescriptionContext: 8192 total in 1 blocks; 6896 free (0 chunks); 1296 used
  MessageContext: 16384 total in 2 blocks; 6648 free (1 chunks); 9736 used
  Operator class cache: 8192 total in 1 blocks; 560 free (0 chunks); 7632 used
  smgr relation table: 16384 total in 2 blocks; 4600 free (2 chunks); 11784 used
  TransactionAbortContext: 32768 total in 1 blocks; 32512 free (0 chunks); 256 used
  Portal hash: 8192 total in 1
  TopPortalContext: 8192 total
    PortalContext: 1024 total
      ExecutorState: 8192 to                                          ed
        printtup: 8192 t
        ExprContext: 8192 total in 1 blocks; 7904 free (0 chunks); 288 used
  Relcache by OID: 16384 total in 2 blocks; 4552 free (2 chunks); 11832 used
  CacheMemoryContext: 524288 total in 7 blocks; 11656 free (1 chunks); 512632 used
    index info: 2048 total in 2 blocks; 728 free (2 chunks); 1320 used: pg_db_role_setting_databas
    index info: 2048 total in 2 blocks; 728 free (2 chunks); 1320 used: pg_user_mapping_user_serve
    index info: 1024 total in 1 blocks; 48 free (0 chunks); 976 used: pg_user_mapping_oid_index
...
    index info: 1024 total in 1 blocks; 48 free (0 chunks); 976 used: pg_ts_parser_oid_index
    index info: 2048 total in 2 blocks; 680 free (1 chunks); 1368 used: pg_attribute_relid_attnum_
    index info: 2048 total in 2 blocks; 952 free (1 chunks); 1096 used: pg_class_oid_index
  WAL record construction: 49768 total in 2 blocks; 6368 free (0 chunks); 43400 used
  PrivateRefCount: 8192 total in 1 blocks; 2624 free (0 chunks); 5568 used
  MdSmgr: 8192 total in 1 blocks; 7336 free (0 chunks); 856 used
  LOCALLOCK hash: 8192 total in 1 blocks; 560 free (0 chunks); 7632 used
  Timezones: 104120 total in 2 blocks; 2624 free (0 chunks); 101496 used
  ErrorContext: 8192 total in 1 blocks; 7936 free (0 chunks); 256 used
Grand total: 1066920 bytes in 158 blocks; 180008 free (75 chunks); 886912 used
```

palloc() happened in this context

Memory context lifetime

- Most code runs in a suitable memory context already

- In special cases, you have to use a longer-lived memory context:
  - Keeping state across function calls
  - Caching something for a transaction or session
    - TransactionContext, TopTransactionContext or TopMemoryContext

Creating your own MemoryContext

- for shorter-lifespan stuff
- for accounting, if you do lots of allocations

```
/*
 * load_tzoffsets --- read and parse the specified timezone offset file
 * ...
 */
TimeZoneAbbrevTable *
load_tzoffsets(const char *filename)
{
    TimeZoneAbbrevTable *result = NULL;
    MemoryContext tmpContext;
    MemoryContext oldContext;
    tzEntry    *array;
    int         arraysize;
    int         n;

    /*
     * Create a temp memory context to work in.  This makes it easy to clean
     * up afterwards.
     */
    tmpContext = AllocSetContextCreate(CurrentMemoryContext,
                                       "TZParserMemory",
                                       ALLOCSET_SMALL_SIZES);
    oldContext = MemoryContextSwitchTo(tmpContext);

    /* Initialize array at a reasonable size */
    arraysize = 128;
    array = (tzEntry *) palloc(arraysize * sizeof(tzEntry));

    /* Parse the file(s) */
    n = ParseTzFile(filename, 0, &array, &arraysize, 0);

    /* If no errors so far, let datetime.c allocate memory & convert format */
    if (n >= 0)
        result = ConvertTimeZoneAbbrevs(array, n);

    /* Clean up */
    MemoryContextSwitchTo(oldContext);
    MemoryContextDelete(tmpContext);

    return result;
}
```

```c
/*
 * load_tzoffsets --- read and parse the specified timezone offset file
 * ...
 */
TimeZoneAbbrevTable *
load_tzoffsets(const char *filename)
{
    TimeZoneAbbrevTable *result = NULL;
    MemoryContext tmpContext;
    MemoryContext oldContext;
    tzEntry    *array;
    int         arraysize;
    int         n;

    /*
     * Create a temp memory context to work in.  This makes it easy to clean
     * up afterwards.
     */
    tmpContext = AllocSetContextCreate(CurrentMemoryContext,
                                       "TZParserMemory",
                                       ALLOCSET_SMALL_SIZES);
    oldContext = MemoryContextSwitchTo(tmpContext);

    /* Initialize array at a reasonable size */
    arraysize = 128;
    array = (tzEntry *) palloc(arraysize * sizeof(

    /* Parse the file(s) */
    n = ParseTzFile(filename, 0, &array, &arraysize, 0);

    /* If no errors so far, let datetime.c allocate memory & convert format */
    if (n >= 0)
        result = ConvertTimeZoneAbbrevs(array, n);

    /* Clean up */
    MemoryContextSwitchTo(oldContext);
    MemoryContextDelete(tmpContext);

    return result;
}
```

Create a new memory context

Leak junk in the context

Copy the result to longer-lived context

Free the junk

String handling

StringInfo

- Expandable buffer for constructing strings

```
/*
 * ExportSnapshot
 *     Export the snapshot to a file so that other backends can import it.
 *         ...
 */
char *
ExportSnapshot(Snapshot snapshot)
{
    ...
    StringInfoData buf;

    /*
     * Fill buf with a text serialization of the snapshot, plus identification
     * data about this transaction.
     */
    initStringInfo(&buf);

    appendStringInfo(&buf, "vxid:%d/%u\n", MyProc->backendId, MyProc->lxid);
    appendStringInfo(&buf, "pid:%d\n", MyProcPid);
    appendStringInfo(&buf, "dbid:%u\n", MyDatabaseId);
    appendStringInfo(&buf, "iso:%d\n", XactIsoLevel);
    appendStringInfo(&buf, "ro:%d\n", XactReadOnly);

    appendStringInfo(&buf, "xmin:%u\n", snapshot->xmin);
    appendStringInfo(&buf, "xmax:%u\n", snapshot->xmax);

    ...

    if (fwrite(buf.data, buf.len, 1, f) != 1)
        ereport(ERROR,
                (errcode_for_file_access(),
                 errmsg("could not write to file \"%s\": %m", pathtmp)));

    ...
```

```
/*
 * ExportSnapshot
 *     Export the snapshot to a file so that other backends can import it.
 *         ...
 */
char *
ExportSnapshot(Snapshot snapshot)
{
    ...
    StringInfoData buf;

    /*
     * Fill buf with a text seri[illegible]tion of the snapshot, plus identification
     * data about this transa[illegible]ion.
     */
    initStringInfo(&buf);

    appendStringInfo(&buf, "vxid:%d/%u\n", MyProc->backendId, MyProc->lxid);
    appendStringInfo(&buf, "pid:%d\n", MyProcPid);
    appendStringInfo(&buf, "dbid:%u\n", MyDatabaseId);
    appendStringInfo(&buf, "iso:%d\n", XactIsoLevel);
    appendStringInfo(&buf, "ro:%d\n", XactReadOnly);

    appendStringInfo(&buf, "xmin:%u\n", snapshot->xmin);
    appendStringInfo(&buf, "xmax:%u\n", s[illegible]

    ...

    if (fwrite(buf.data, buf.len, 1, f) != 1)
        ereport(ERROR,
                (errcode_for_file_access(),
                 errmsg("could not write to file \"%s\": %m", pathtmp)));

    ...
```

Initialize new buffer

Append stuff

Result is in buf.data and buf.len

StringInfo

- Can also be used as an expandable buffer for binary data
  - appendBinaryStringInfo(<buf>, <data>, <len>)

Other string functions

- pstrdup()
- psprintf()

- pg_strtouint64()
- pg_strcasecmp() and pg_strncasecmp()
- pg_tolower()/pg_toupper()
- pg_strtok()
- pg_strftime()

Building SQL strings

```
/* First fetch Oid and replica identity. */
initStringInfo(&cmd);
appendStringInfo(&cmd,
    "SELECT c.oid, c.relreplident"
    "  FROM pg_catalog.pg_class c"
    "  INNER JOIN pg_catalog.pg_namespace n"
    "        ON (c.relnamespace = n.oid)"
    " WHERE n.nspname = %s"
    "   AND c.relname = %s"
    "   AND c.relkind = 'r'",
    quote_literal_cstr(nspname),
    quote_literal_cstr(relname));
```

Building SQL strings

```
/* First fetch Oid and replica identity. */
initStringInfo(&cmd);
appendStringInfo(&cmd,
     "SELECT c.oid, c.relre
     "  FROM pg_catalog.pg_class c"
     "  INNER JOIN pg_catalog.pg_namespace n"
     "        ON (c.relnamespace = n.oid)"
     " WHERE n.nspname = %s"
     "   AND c.relname = %s"
     "   AND c.relkind = 'r'",
     quote_literal_cstr(nspname),
     quote_literal_cstr(relname));
```

Always schema-qualify built-in tables and functions

Remember quoting!

Building SQL strings

- quote_identifier()
  - “SELECT * FROM %s”, quote_identifier(tablename)
- quote_literal_cstr()
  - “SELECT * FROM table WHERE column = %s”, quote_literal_cstr()

- Check the examples in user manual on how to use SPI

Error handling

ereport(ERROR, ...)

```
/*
 *    dsqrt     - returns square root of arg1
 */
Datum
dsqrt(PG_FUNCTION_ARGS)
{
    float8   arg1 = PG_GETARG_FLOAT8(0);
    float8   result;

    if (arg1 < 0)
        ereport(ERROR,
                (errcode(ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION),
                 errmsg("cannot take square root of a negative number")));

    result = sqrt(arg1);

    check_float8_val(result, isinf(arg1), arg1 == 0);
    PG_RETURN_FLOAT8(result);
}
```

ereport(ERROR, ...)

```
/*
 *    dsqrt     - returns square root of arg1
 */
Datum
dsqrt(PG_FUNCTION_ARGS)
{
    float8   arg1 = PG_GETARG_FLOAT8(0);
    float8   result;

    if (arg1 < 0)
        ereport(ERROR,
                (errcode(ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION),
                 errmsg("cannot take square root of a negative number")));

    result = sqrt(arg1

    check_floa
    PG_RETURN_
}
```

Execution stops here with ERROR

```
psql (12devel)
Type "help" for help.

postgres=# select sqrt(-1);
ERROR:  cannot take square root of a negative number
postgres=#
```

Error levels

- ERROR
  - Abort transaction
  - Return to main query loop
- FATAL
  - Disconnect the client, terminate the process gracefully.
- PANIC
  - Immediately shut down the server, enter crash recovery.

Non-error levels

- LOG
  - Printed to log
- WARNING
- NOTICE
- INFO
- DEBUG1-5

Error *codes*

```
ereport(ERROR,
        (errcode(ERRCODE_INVALID_ARGUMENT_FOR_POWER_FUNCTION),
         errmsg("cannot take square root of a negative number")));
```

- Sent to client as “SQLSTATE”
- Default is ERRCODE_INTERNAL_ERROR
- Can be used in application to distinguish certain errors
  - e.g. serialization failure → retry
  - unique key violation
- List in src/backend/utils/errcodes.txt

Error *codes*

```
psql (12devel)
Type "help" for help.

postgres=# \set VERBOSITY verbose
postgres=# select sqrt(-1);
ERROR:  2201F: cannot take square root of a negative number
LOCATION:  dsqrt, float.c:1323
postgres=#
```

Catching exceptions

- PG_TRY – PG_CATCH
- The error MUST be re-thrown!

```c
/*
 * Convert XML node to text (dump subtree in case of element,
 * return value otherwise)
 */
static text *
xml_xmlnodetoxmltype(xmlNodePtr cur, PgXmlErrorContext *xmlerrcxt)
{
    xmltype    *result;

    if (cur->type == XML_ELEMENT_NODE)
    {
        xmlBufferPtr buf;
        xmlNodePtr    cur_copy;

        buf = xmlBufferCreate();

        /*
         * The result of xmlNodeDump() won't contain namespace definitions
         * from parent nodes, but xmlCopyNode() duplicates a node along with
         * its required namespace definitions.
         */
        cur_copy = xmlCopyNode(cur, 1);

        if (cur_copy == NULL)
            xml_ereport(xmlerrcxt, ERROR, ERRCODE_OUT_OF_MEMORY,
                        "could not copy node");

        PG_TRY();
        {
            xmlNodeDump(buf, NULL, cur_copy, 0, 1);
            result = xmlBuffer_to_xmltype(buf);
        }
        PG_CATCH();
        {
            xmlFreeNode(cur_copy);
            xmlBufferFree(buf);
            PG_RE_THROW();
        }
        PG_END_TRY();
        xmlFreeNode(cur_copy);
        xmlBufferFree(buf);
    }
    Else
    {
        ...
```

```
buf = xmlBufferCreate();

/*
 * The result of xmlNodeDump() won't contain namespace definitions
 * from parent nodes, but xmlCopyNode() duplicates a node along with
 * its required namespace definitions.
 */
cur_copy = xmlCopyNode(cur, 1);

if (cur_copy == NULL)
    xml_ereport(xmlerrcxt, ERROR, ERRCODE_OUT_OF_MEMORY,
                "could not copy node");

PG_TRY();
{
    xmlNodeDump(buf, NULL, cur_copy, 0, 1);
    result = xmlBuffer_to_xmltype(buf);
}
PG_CATCH();
{
    xmlFreeNode(cur_copy);
    xmlBufferFree(buf);
    PG_RE_THROW();
}
PG_END_TRY();
xmlFreeNode(cur_copy);
xmlBufferFree(buf);
```

libxml call allocates stuff

libxml call allocates stuff

PostgreSQL functions that might ereport(ERROR)

Free libxml releases on error

Critical sections

```
static void
brin_initialize_empty_new_buffer(Relation idxrel, Buffer buffer)
{
    Page        page;

    elog(DEBUG2,
         "brin_initialize_empty_new_buffer: initializing blank page %u",
         BufferGetBlockNumber(buffer)));

    START_CRIT_SECTION();
    page = BufferGetPage(buffer);
    brin_page_init(page, BRIN_PAGETYPE_REGULAR);
    MarkBufferDirty(buffer);
    log_newpage_buffer(buffer, true);
    END_CRIT_SECTION();

    /*
     * We update the FSM for this page, but this is not WAL-logged.  This is
     * acceptable because VACUUM will scan the index and update the FSM with
     * pages whose FSM records were forgotten in a crash.
     */
    RecordPageWithFreeSpace(idxrel, BufferGetBlockNumber(buffer),
    br_page_get_freespace(page));
}
```

Critical sections

```
static void
brin_initialize_empty_new_buffer(Rel
{
    Page        page;

    elog(DEBUG2,
         "brin_initialize_empty_new_
         BufferGetBlockNumber(buffer),

    START_CRIT_SECTION();
    page = BufferGetPage(buffer);
    brin_page_init(page, BRIN_PAGETYPE_REGULAR);
    MarkBufferDirty(buffer);
    log_newpage_buffer(buffer, true);
    END_CRIT_SECTION();

    /*
     * We update the FSM for this page, but this is not WAL-logged.  This is
     * acceptable because VACUUM will scan the index and update the FSM with
     * pages whose FSM records were forgotten in a crash.
     */
    RecordPageWithFreeSpace(idxrel, BufferGetBlockNumber(buffer),
    br_page_get_freespace(page));
}
```

Any ERROR within critical section will be promoted to **PANIC**, and leads to immediate shutdown and crash recovery.

Resource Owners

Accessing files

- fopen() -> AllocateFile()
- open() → BasicOpenFile()

- Closed automatically at end of transaction
- Manage running out of file descriptors

```
if ((file = AllocateFile(filename, PG_BINARY_R)) == NULL)
{
    if (missing_ok && errno == ENOENT)
        return NULL;
    else
        ereport(ERROR,
                (errcode_for_file_access(),
                 errmsg("could not open file \"%s\" for reading: %m",
                        filename)));
}

if (fseeko(file, (off_t) seek_offset, (seek_offset >= 0) ? SEEK_SET : SEEK_END) != 0)
    ereport(ERROR,
            (errcode_for_file_access(),
             errmsg("could not seek in file \"%s\": %m", filename)));

buf = (bytea *) palloc((Size) bytes_to_read + VARHDRSZ);

nbytes = fread(VARDATA(buf), 1, (size_t) bytes_to_read, file);

if (ferror(file))
    ereport(ERROR,
            (errcode_for_file_access(),
             errmsg("could not read file \"%s\": %m", filename)));

SET_VARSIZE(buf, nbytes + VARHDRSZ);

Freefile(file);

return buf;
```

```
if ((file = AllocateFile(filename, PG_BINARY_R)) == NULL)
{
    if (missing_ok && er
        return NULL;
    else
        ereport(ERROR,
                (errcode_for_file_access(),
                 errmsg("could not open file \"%s\" for reading: %m",
                        filename)));
}

if (fseeko(file, (off_t) seek_offset, (seek_offset >= 0) ? SEEK_SET : SEEK_END) != 0)
    ereport(ERROR,
            (errcode_for_file_access(),
             errmsg("could not seek in

buf = (bytea *) palloc((Size) bytes_to_r

nbytes = fread(VARDATA(buf), 1, (size_t)      _to_read, file);

if (ferror(file))
    ereport(ERROR,
            (errcode_for_file_access(),
             errmsg("could not read file \"%s\": %m", filename)));

SET_VARSIZE(buf, nbytes + VARHDRSZ);

Freefile(file);

return buf;
```

Open a file, like fopen() does

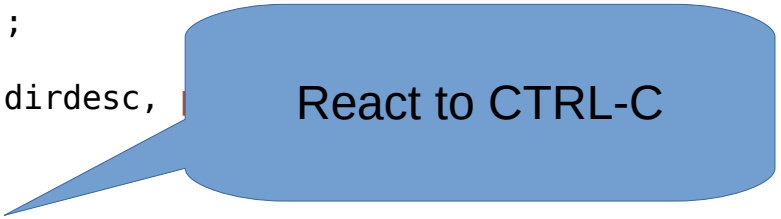
On ERROR, the file will be automatically closed at rollback

Walking directories

- Like with files
- AllocateDir()
- ReadDir()
- Tracked by resource owner

```c
/* Return physical size of directory contents, or 0 if dir doesn't exist */
static int64
db_dir_size(const char *path)
{
    int64      dirsize = 0;
    struct dirent *direntry;
    DIR       *dirdesc;
    char       filename[MAXPGPATH * 2];

    dirdesc = AllocateDir(path);

    while ((direntry = ReadDir(dirdesc, path)) != NULL)
    {
        struct stat fst;

        CHECK_FOR_INTERRUPTS();

        if (strcmp(direntry->d_name, ".") == 0 ||
            strcmp(direntry->d_name, "..") == 0)
            continue;

        snprintf(filename, sizeof(filename), "%s/%s", path, direntry->d_name);

        if (stat(filename, &fst) < 0)
        {
            if (errno == ENOENT)
                continue;
            else
                ereport(ERROR,
                        (errcode_for_file_access(),
                         errmsg("could not stat file \"%s\": %m", filename)));
        }
        dirsize += fst.st_size;
    }

    FreeDir(dirdesc);
    return dirsize;
}
```

```c
/* Return physical size of directory contents, or 0 if dir doesn't exist */
static int64
db_dir_size(const char *path)
{
    int64       dirsize = 0;
    struct dirent *direntry;
    DIR        *dirdesc;
    char        filename[MAXPGPATH * 2];

    dirdesc = AllocateDir(path);

    while ((direntry = ReadDir(dirdesc, 
    {
        struct stat fst;

        CHECK_FOR_INTERRUPTS();

        if (strcmp(direntry->d_name, ".") == 0 ||
            strcmp(direntry->d_name, "..") == 0)
            continue;

        snprintf(filename, sizeof(filename), "%s/%s", path, direntry->d_name);

        if (stat(filename, &fst) < 0)
        {
            if (errno == ENOENT)
                continue;
            else
                ereport(ERROR,
                        (errcode_for_file_access(),
                         errmsg("could not stat file \"%s\": %m", filename)));
        }
        dirsize += fst.st_size;
    }

    FreeDir(dirdesc);
    return dirsize;
}
```



Interrupts, or responding to CTRL-C

- CHECK_FOR_INTERRUPTS()
- Also checked by many internal functions
- Holding a LWLock prevents interrupt

Resource Owners

- Files are tracked by ResourceOwners
- Hierarchy similar to MemoryContexts
- Track files, relcache references, locks, buffer pins, etc.

Data structures

In-memory data structures

- Linked lists (single- and doubly-linked)
- Hash tables
- Binary and pairing heaps
- Red-black tree
- Hyperloglog

Sorting and buffering

- Tuplesort.c
- Tuplestore.c

- These spill to disk

Hash tables - dynahash

```
/*
 * We use a hash table to hold known names, so that this process is O(N)
 * not O(N^2) for N names.
 */
MemSet(&hash_ctl, 0, sizeof(hash_ctl));
hash_ctl.keysize = NAMEDATALEN;
hash_ctl.entrysize = sizeof(NameHashEntry);
hash_ctl.hcxt = CurrentMemoryContext;
names_hash = hash_create("set_rtable_names names",
                         list_length(dpns->rtable),
                         &hash_ctl,
                         HASH_ELEM | HASH_CONTEXT);

/* Preload the hash table with names appearing in parent_namespaces */
foreach(lc, parent_namespaces)
{
    deparse_namespace *olddpns = (deparse_namespace *) lfirst(lc);
    ListCell   *lc2;

    foreach(lc2, olddpns->rtable_names)
    {
        char    *oldname = (char *) lfirst(lc2);

        if (oldname == NULL)
            continue;
        hentry = (NameHashEntry *) hash_search(names_hash,
                                               oldname,
                                               HASH_ENTER,
                                               &found);
        /* we do not complain about duplicate names in parent namespaces */
        hentry->counter = 0;
    }
}
```

Hash tables - dynahash

```
/*
 * We use a hash table to hold known names, so that this process is O(N)
 * not O(N^2) for N names.
 */
MemSet(&hash_ctl, 0, sizeof(hash_ctl));
hash_ctl.keysize = NAMEDATALEN;
hash_ctl.entrysize = sizeof(NameHashEntry);
hash_ctl.hcxt = CurrentMemoryContext;
names_hash = hash_create("set_rtable_names names",
                         list_length(dpns->rtable),
                         &hash_ctl,
                         HASH_ELEM | HASH_CONTEXT);

/* Preload the hash table with names appearing in parent_namespaces */
foreach(lc, parent_namespaces)
{
    deparse_namespace *olddpns = (deparse_namespace *) lfirst(lc);
    ListCell   *lc2;

    foreach(lc2, olddpns->rtable_names)
    {
        char    *oldname = (char *) lfirst(lc2);

        if (oldname == NULL)
            continue;
        hentry = (NameHashEntry *) hash_search(names_hash,
                                               oldname,
                                               HASH_ENTER,
                                               &found);
        /* we do not complain about duplicate names in parent namespaces */
        hentry->counter = 0;
    }
}
```

Creation:
1. fill a struct with options
2. call hash_create

hash_search() to add and search for entries

Dynahash

- Backend-local or shared memory
- Works with Memory Contexts
- Flexible

Hash tables - “simple hash”

- simplehash.h
- Robin hood hashing
- Consists of macros that are inlined
- Fast

Linked lists

pg_list.h

- List, ListCell
- lfirst(), lnext()
- Foreach()

ilist.h

- embedded in other structs
- dlist_node, dlist_iter, dlist_foreach()
- slit_node, dlist_iter, etc.

Inter-process communication

- PostgreSQL is a multi-process system
- Locking
  - LWLocks
  - Heavy-weight locks
  - Predicate locks
  - Spinlocks
  - Atomic CPU instructions
  - Memory barriers
- Shared memory

What **not** to do

- malloc/free (use MemoryContexts)
- Open files (use BasicOpenFile() etc)
- Signal handlers
- Fork (background workers)
- Launch threads (background workers)
- C++ exceptions (ereport())

Tips & Tricks

- Attach a debugger to backend process
  - SELECT pg_backend_pid()
  - gdb - <pid>
- ‘rr’ reverse debugger works on PostgreSQL
  - Launch server with “rr record postgres -D ...”
  - Run query
  - Stop server
  - Run “rr replay -f <pid>”

How do I do X?

- Search the source code for examples
  - Keep the PostgreSQL sources at hand
- Search the git commit history for the last time that was done
- Careful if you copy-paste random stuff from the Internet
  - lots of old code
- Search the mailing list archives, post a question
- IRC

The end

Questions?