Архитектура

Буферный кеш и журнал

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Игорь Гнатюк
Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Устройство буферного кеша

Алгоритм вытеснения

Журнал предзаписи

Контрольная точка

Процессы, связанные с буферным кешем и журналом

Буферный кеш

Массив буферов

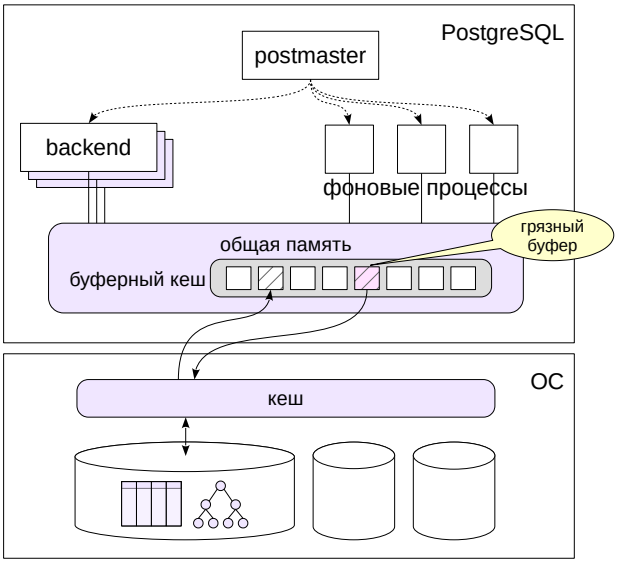
- страница данных (8 Кбайт)
- доп. информация

«Грязные» буферы

- асинхронная запись

Блокировки в памяти

- для совместного доступа



Буферный кеш используется для сглаживания скорости работы оперативной памяти и дисков. Он состоит из массива буферов, которые содержат страницы данных и дополнительную информацию (например, имя файла и положение страницы внутри этого файла).

Размер страницы обычно составляет 8 Кбайт; размер можно изменить только при сборке PostgreSQL.

Любая работа со страницами данных проходит через буферный кеш. Если какой-либо процесс собирается работать со страницей, он в первую очередь пытается найти ее в кеше. Если страницы нет, он обращается к операционной системе с просьбой прочитать эту страницу и помещает ее в буферный кеш. (Обратите внимание, что ОС может прочитать страницу с диска, а может обнаружить ее в собственном кеше.)

После того, как страница записана в буферный кеш, к ней можно обращаться многократно без накладных расходов на вызовы ОС.

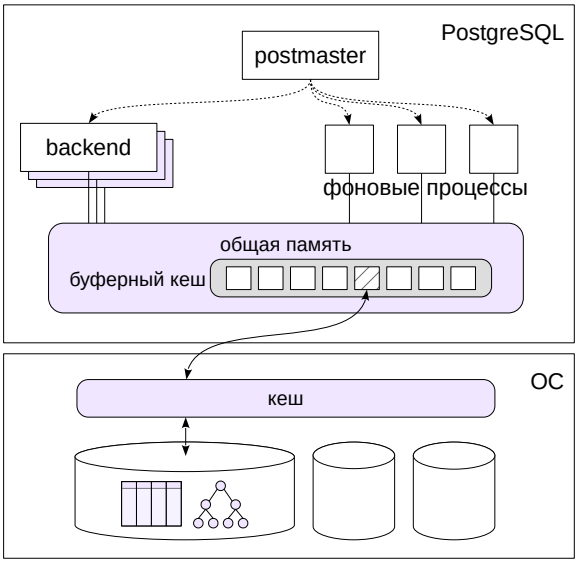
Если процесс изменил данные в странице, соответствующий буфер становится «грязным». Измененная страница подлежит записи на диск, но по соображениям производительности запись происходит асинхронно и может откладываться.

Буферный кеш, как и другие структуры общей памяти, защищен блокировками для управления одновременным доступом. Хотя блокировки и реализованы эффективно, доступ к буферному кешу далеко не так быстр, как простое обращение к оперативной памяти. Поэтому в общем случае чем меньше данных читает и изменяет запрос, тем быстрее он будет работать.

Вытеснение

Вытеснение редко используемых страниц

- грязный буфер записывается на диск
- на освободившееся место читается другая страница



Размер буферного кеша обычно не так велик, чтобы база данных помещалась в него целиком. Его ограничивают и доступная оперативная память, и возрастающие при его увеличении накладные расходы. Поэтому при чтении очередной страницы рано или поздно окажется, что место в буферном кеше закончилось. В этом случае применяется *вытеснение* страниц.

Алгоритм вытеснения выбирает в кеше страницу, которая в последнее время использовалась реже других. Если выбранный буфер оказался грязным, страница записывается на диск, чтобы не потерять сделанные в ней изменения. Затем в освободившийся буфер читается новая страница.

Такой алгоритм вытеснения называется LRU — Least Recently Used. Он сохраняет в кеше данные, с которыми происходит активная работа. Таких «горячих» данных обычно не слишком много, и при достаточном объеме буферного кеша получается существенно сократить количество обращений к ОС (и дисковых операций).

Влияние буферного кеша на выполнение запросов

Создаем новую БД в кластере и подключаемся к ней (подробнее про базы данных — в модуле «Организация данных»):

```
=> CREATE DATABASE arch_wal_overview;

CREATE DATABASE

=> \c arch_wal_overview

You are now connected to database "arch_wal_overview" as user "student".
```

Создадим таблицу:

```
=> CREATE TABLE t(n integer);

CREATE TABLE
```

Заполним таблицу некоторым количеством строк:

```
=> INSERT INTO t SELECT id FROM generate_series(1,100_000) AS id;

INSERT 0 100000
```

Размер буферного кеша показывает параметр shared_buffers:

```
=> SHOW shared_buffers;

 shared_buffers
----------------
 128MB
(1 row)
```

Значение по умолчанию слишком мало; в любой реальной системе его требуется увеличить сразу после установки сервера (изменение требует перезапуска).

Теперь перезапустим сервер, чтобы содержимое буферного кеша сбросилось.

```
=> \q

student$ sudo pg_ctlcluster 16 main restart

student$ psql arch_wal_overview
```

Сравним, что происходит при первом и при втором выполнении одного и того же запроса. В этом курсе мы не рассматриваем подробно планы запросов, но иногда будем в них заглядывать. Сейчас мы воспользуемся командой EXPLAIN ANALYZE, которая выполняет запрос и выводит не только план выполнения, но и дополнительную информацию:

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
SELECT * FROM t;

                QUERY PLAN
-------------------------------------------
 Seq Scan on t (actual rows=100000 loops=1)
   Buffers: shared read=443
 Planning:
   Buffers: shared hit=12 read=8 dirtied=1
 Planning Time: 0.185 ms
 Execution Time: 7.796 ms
(6 rows)
```

Строка «Buffers: shared» показывает использование буферного кеша.

- read — количество буферов, в которые пришлось прочитать страницы с диска.

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
SELECT * FROM t;

                QUERY PLAN
-------------------------------------------
 Seq Scan on t (actual rows=100000 loops=1)
   Buffers: shared hit=443
 Planning Time: 0.029 ms
 Execution Time: 5.180 ms
(4 rows)
```

- hit — количество буферов, в которых нашлись нужные для запроса страницы.

Обратите внимание, что во второй раз уменьшилось и время выполнения запроса, и время его планирования (потому что таблицы системного каталога тоже кешируются).

Журнал предзаписи (WAL)

Проблема: при сбое теряются данные из оперативной памяти, не записанные на диск

Журнал

- поток информации о выполняемых действиях, позволяющий повторно выполнить потерянные при сбое операции
- запись попадает на диск раньше, чем измененные данные

Журнал защищает

- страницы таблиц, индексов и других объектов
- статус транзакций (clog)

Журнал не защищает

- временные и нежурналируемые таблицы

Наличие буферного кеша (и других буферов в оперативной памяти) увеличивает производительность, но уменьшает надежность.
В случае сбоя в СУБД содержимое буферного кеша потеряется. Если сбой произойдет в операционной системе или на аппаратном уровне, то пропадет содержимое и буферов ОС (но с этим справляется сама операционная система).

Для обеспечения надежности PostgreSQL использует журналирование. При выполнении любой операции формируется запись, содержащая минимально необходимую информацию для того, чтобы операцию можно было выполнить повторно. Такая запись должна попасть на диск (или другой энергонезависимый накопитель) раньше, чем будут записаны изменяемые операцией данные (поэтому журнал и называется *журналом предзаписи*, write-ahead log).

Файлы журнала располагаются в каталоге PGDATA/pg_wal.

Журнал защищает все объекты, работа с которыми ведется в оперативной памяти: таблицы, индексы и другие объекты, статус транзакций.

В журнал не попадают данные о *временных таблицах* (такие таблицы доступны только создавшему их пользователю и только на время сеанса или транзакции) и о *нежурналируемых таблицах* (такие таблицы ничем не отличаются от обычных, кроме того, что не защищены журналом). В случае сбоя нежурналируемые таблицы просто очищаются, зато работа с ними происходит быстрее.

https://postgrespro.ru/docs/postgresql/16/wal-intro

Журнал предзаписи

Логически журнал можно представить в виде непрерывного потока записей. Каждая запись имеет номер, называемый LSN (Log Sequence Number). Это 64-разрядное число — смещение записи в байтах относительно начала журнала.

Текущую позицию показывает функция pg_current_wal_lsn:

```
=> SELECT pg_current_wal_lsn();

 pg_current_wal_lsn
--------------------
 0/2367068
(1 row)
```

Позиция записывается как два 32-разрядных числа через косую черту. Запомним это значение.

Выполним теперь какие-нибудь операции и посмотрим, как изменилась позиция.

```
=> UPDATE t SET n = 100_001 WHERE n = 1;

UPDATE 1

=> SELECT pg_current_wal_lsn();

 pg_current_wal_lsn
--------------------
 0/236A098
(1 row)
```

Интересны не абсолютные числа, а их разница, которая показывает размер сгенерированных журнальных записей в байтах:

```
=> SELECT '0/236A098'::pg_lsn - '0/2367068'::pg_lsn AS bytes;

 bytes
-------
 12336
(1 row)
```

Физически журнал хранится в файлах в отдельном каталоге (PGDATA/pg_wal). По умолчанию файлы имеют размер 16 Мбайт, задать другой размер можно при инициализации кластера баз данных.

На файлы можно взглянуть не только средствами файловой системы, но и с помощью функции:

```
=> SELECT * FROM pg_ls_waldir() ORDER BY name LIMIT 10;

           name           |   size   |      modification
--------------------------+----------+------------------------
 000000010000000000000002 | 16777216 | 2024-07-05 16:18:51+03
 000000010000000000000003 | 16777216 | 2024-07-05 16:18:43+03
(2 rows)
```

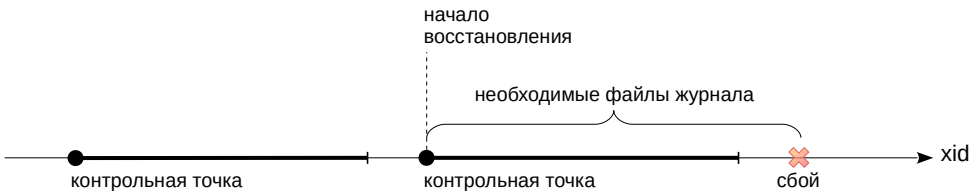
Контрольная точка

Периодический сброс всех грязных буферов на диск

- гарантирует попадание на диск всех изменений до контрольной точки
- ограничивает размер журнала, необходимого для восстановления

Восстановление при сбое

- начинается с последней контрольной точки
- последовательно проигрываются записи, если изменений нет на диске



Когда сервер PostgreSQL запускается после сбоя, он входит в режим восстановления. Информация на диске в это время несогласованна: изменения «горячих» страниц пропали, поскольку эти страницы все еще находились в кеше, а более поздние изменения уже были сброшены на диск.

Чтобы восстановить согласованность, PostgreSQL читает WAL и последовательно проигрывает каждую его запись, если соответствующее изменение не попало на диск. Таким образом восстанавливается работа всех транзакций. Транзакции, запись о фиксации которых не успела попасть в журнал, считаются оборванными.

Однако объем журнала за время работы сервера мог бы достигнуть гигантских размеров. Хранить его целиком и просматривать при сбое совершенно не реально. Поэтому СУБД периодически выполняет контрольную точку (КТ): принудительно сбрасывает на диск все грязные буферы (включая буферы clog, хранящие состояние транзакций).

Собственно «точка» — это момент начала записи буферов, которые были грязными на этот момент. Но точка считается выполненной только после того, как все такие буферы записаны. Это гарантирует, что все изменения, сделанные до момента КТ, находятся на диске.

В реальных системах с большим буферным кешем КТ может сбрасывать много грязных буферов, поэтому сервер распределяет процесс записи по времени, чтобы сгладить нагрузку на ввод-вывод.

Восстановление после сбоя начинается с момента последней завершенной КТ, что позволяет хранить только файлы журнала, записанные после нее.

Восстановление при помощи журнала

Измененные табличные страницы находятся в буферном кеше, но еще не записаны на диск. При обычной остановке сервер выполняет контрольную точку, чтобы записать все грязные страницы на диск, но мы сымитируем сбой системы, послав сигнал процессу postmaster.

```
student$ sudo head -n 1 /var/lib/postgresql/16/main/postmaster.pid
```

```
8215
```

```
student$ sudo kill -QUIT 8215
```

При старте произойдет восстановление согласованности данных с помощью журнала. Проверим:

```
student$ sudo pg_ctlcluster 16 main start
```

```
student$ psql arch_wal_overview
```

```
=> SELECT min(n), max(n) FROM t;
```

```
 min |  max
-----+--------
   2 | 100001
(1 row)
```

Все изменения были восстановлены.

PostgreSQL автоматически удаляет журнальные файлы, не требующиеся для восстановления, после выполнения контрольной точки.

Производительность

Синхронный режим

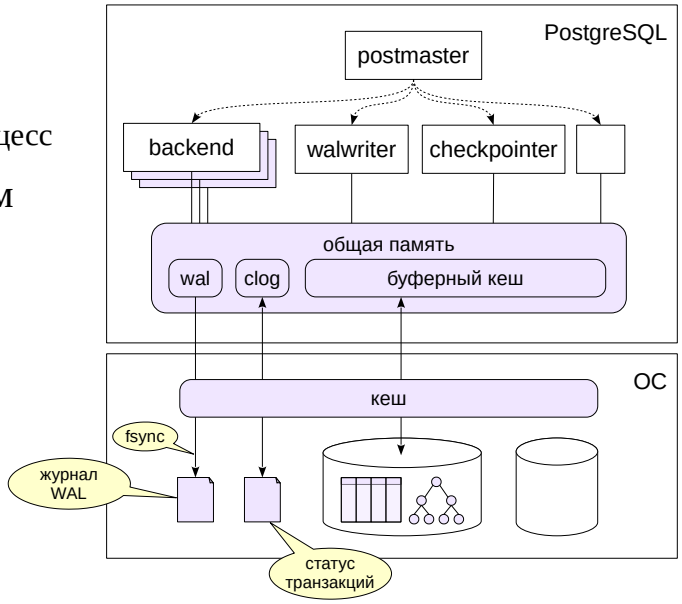
запись при фиксации

обслуживающий процесс

Асинхронный режим

фоновая запись

walwriter



Механизм журналирования более эффективен, чем работа напрямую с диском без буферного кеша. Во-первых, размер журнальных записей меньше, чем размер целой страницы данных; во-вторых, журнал записывается строго последовательно (и обычно не читается, пока не случится сбой), с чем вполне справляются простые HDD-диски.

На эффективность можно также влиять настройкой. Если запись происходит сразу (синхронно), то гарантируется, что зафиксированная транзакция не пропадет. Но запись — довольно дорогая операция, в течение которой обслуживающий процесс, выполняющий фиксацию, вынужден ждать. Чтобы журнальная запись не «застряла» в кеше операционной системы, выполняется системный вызов fsync: PostgreSQL полагается на то, что это гарантирует попадание данных на энергонезависимый носитель.

Поэтому есть и режим отложенной (асинхронной) записи. В этом случае записи пишутся фоновым процессом walwriter постепенно, с небольшой задержкой. Надежность уменьшается, зато производительность увеличивается. Но и в этом случае после сбоя гарантируется восстановление согласованности.

На самом деле оба режима работают совместно. Журнальные записи долгой транзакции будут записываться асинхронно (чтобы освободить буферы WAL). А если при сбросе страницы данных окажется, что соответствующая журнальная запись еще не на диске, она тут же будет записана в синхронном режиме.

Основные процессы

Запись журнала

Контрольная точка

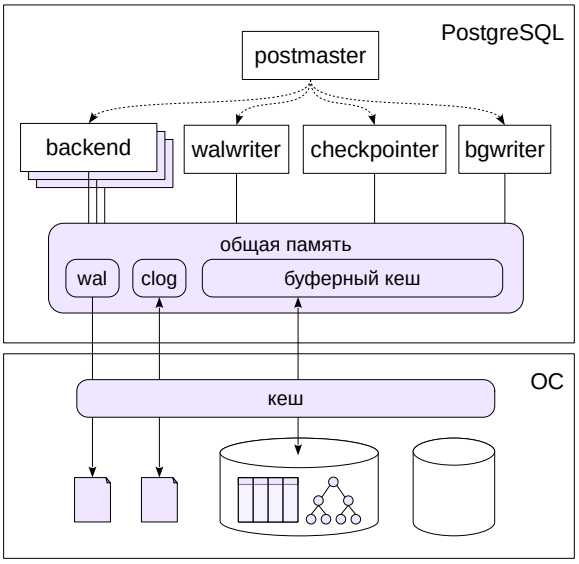
- сброс всех грязных буферов

Фоновая запись

- сброс части грязных буферов

Обслуживающие процессы

- сброс вытесняемого грязного буфера



Вернемся еще раз к процессам, связанным с обслуживанием буферного кеша и журнала.

Во-первых, это процесс walwriter, занимающийся асинхронной записью журнала на диск. При синхронном режиме записью журнала занимается тот процесс, который выполняет фиксацию транзакции.

Во-вторых, процесс контрольной точки checkpointer, периодически сбрасывающий все грязные буферы на диск.

В-третьих, процесс фоновой записи background writer (или bgwriter). Этот процесс похож на процесс контрольной точки, но записывает только часть грязных буферов, причем те, которые с большой вероятностью будут вытеснены в ближайшее время. Таким образом, когда обслуживающий процесс выберет буфер, чтобы прочитать новую страницу, старая страница скорее всего уже не будет грязной и не надо будет тратить время, чтобы сбросить ее на диск.

И в-четвертых, обслуживающие процессы, читающие данные в буферный кеш. Если, несмотря на работу процессов контрольной точки и фоновой записи, вытесняемый буфер окажется грязным, обслуживающий процесс самостоятельно запишет его на диск.

Уровни журнала

Minimal

гарантия восстановления после сбоя

Replica *(по умолчанию)*

резервное копирование

репликация: передача и проигрывание журнала на другом сервере

Logical

логическая репликация: информация о добавлении, изменении и удалении табличных строк

Как уже говорилось, причиной появления журнала является необходимость защищать информацию от сбоев из-за потери содержимого оперативной памяти.

Однако журнал — механизм, который оказалось удобно применять и для других целей, если добавить в него дополнительную информацию.

Объем данных, который попадает в журнал, регулируется параметром *wal_level*.

- На уровне **minimal** журнал обеспечивает только восстановление после сбоя.
- На уровне **replica** в журнал добавляется информация, позволяющая использовать его для создания резервных копий и репликации. При репликации журнальные записи передаются на другой сервер и применяются там; таким образом создается и поддерживается точная копия (реплика) основного сервера.
- На уровне **logical** в журнал добавляется информация, позволяющая декодировать «физические» журнальные записи и сформировать из них «логические» записи о добавлении, изменении и удалении табличных строк. Это позволяет организовать логическую репликацию (рассматривается в соответствующих темах курсов DEV2 и DBA3).

Итоги

Буферный кеш существенно ускоряет работу,
уменьшая число дисковых операций

Надежность обеспечивается журналированием

Размер журнала ограничен благодаря контрольным точкам

Журнал удобен и используется во многих случаях

- для восстановления после сбоя
- при резервном копировании
- для репликации между серверами

Практика

1. Средствами операционной системы найдите процессы, отвечающие за работу буферного кеша и журнала WAL.
2. Остановите PostgreSQL в режиме fast; снова запустите его. Просмотрите журнал сообщений сервера.
3. Теперь остановите в режиме immediate и снова запустите. Просмотрите журнал сообщений сервера и сравните с предыдущим разом.

2. Для останова в режиме fast используйте команду

```
pg_ctlcluster 16 main stop
```

При этом сервер обрывает все открытые соединения и перед выключением выполняет контрольную точку, чтобы на диск записались согласованные данные. Таким образом, выключение может выполняться относительно долго, но при запуске сервер сразу же будет готов к работе.

3. Для останова в режиме immediate используйте команду

```
pg_ctlcluster 16 main stop -m immediate --skip-systemctl-redirect
```

При этом сервер также обрывает открытые соединения, но не выполняет контрольную точку. На диске остаются несогласованные данные, как после сбоя. Таким образом, выключение происходит быстро, но при запуске сервер должен будет восстановить согласованность данных с помощью журнала.

Для PostgreSQL, собранного из исходных кодов, останов в режиме fast выполняется командой

```
pg_ctl stop
```

а останов в режиме immediate — командой

```
pg_ctl stop -m immediate
```

1. Процессы операционной системы

Сначала получим идентификатор процесса postmaster. Он записан в первой строке файла postmaster.pid. Этот файл расположен в каталоге с данными и создается каждый раз при старте сервера.

```
student$ sudo cat /var/lib/postgresql/16/main/postmaster.pid
```

```
47662
/var/lib/postgresql/16/main
1720186158
5432
/var/run/postgresql
localhost
   684212     32774
ready
```

Теперь найдем все процессы, порожденные процессом postmaster:

```
student$ sudo ps -o pid,command --ppid 47662
```

```
    PID COMMAND
  47663 postgres: 16/main: checkpointer
  47664 postgres: 16/main: background writer
  47666 postgres: 16/main: walwriter
  47667 postgres: 16/main: autovacuum launcher
  47668 postgres: 16/main: logical replication launcher
  47709 postgres: 16/main: student student [local] idle
```

К процессам, обслуживающим буферный кеш и журнал, можно отнести:

- checkpointer;
- background writer;
- walwriter.

2. Остановка в режиме fast

Чтобы легко отделить старые сообщения от новых, мы просто удалим журнал сообщений перед перезапуском сервера. Конечно, в реальной работе так поступать не следует.

```
student$ sudo rm /var/log/postgresql/postgresql-16-main.log
```

```
student$ sudo pg_ctlcluster 16 main stop
```

```
student$ sudo pg_ctlcluster 16 main start
```

Журнал сообщений сервера:

```
student$ cat /var/log/postgresql/postgresql-16-main.log
```

```
2024-07-05 16:29:23.662 MSK [47941] LOG:  starting PostgreSQL 16.3 (Ubuntu
16.3-1.pgdg22.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu
11.4.0-1ubuntu1~22.04) 11.4.0, 64-bit
2024-07-05 16:29:23.662 MSK [47941] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2024-07-05 16:29:23.677 MSK [47941] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2024-07-05 16:29:23.722 MSK [47944] LOG:  database system was shut down at 2024-07-05
16:29:23 MSK
2024-07-05 16:29:23.744 MSK [47941] LOG:  database system is ready to accept connections
```

3. Остановка в режиме immediate

```
student$ sudo rm /var/log/postgresql/postgresql-16-main.log
```

```
student$ sudo pg_ctlcluster 16 main stop -m immediate --skip-systemctl-redirect
```

```
student$ sudo pg_ctlcluster 16 main start
```

Журнал сообщений сервера:

```
student$ cat /var/log/postgresql/postgresql-16-main.log
```

```
2024-07-05 16:29:26.422 MSK [48081] LOG:  starting PostgreSQL 16.3 (Ubuntu
16.3-1.pgdg22.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu
11.4.0-1ubuntu1~22.04) 11.4.0, 64-bit
2024-07-05 16:29:26.422 MSK [48081] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2024-07-05 16:29:26.440 MSK [48081] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2024-07-05 16:29:26.478 MSK [48084] LOG:  database system was interrupted; last known up
```

```
at 2024-07-05 16:29:23 MSK
2024-07-05 16:29:36.549 MSK [48084] LOG:  syncing data directory (fsync), elapsed time:
10.04 s, current path: ./base/5/2674
2024-07-05 16:29:46.504 MSK [48084] LOG:  syncing data directory (fsync), elapsed time:
20.00 s, current path: ./base/4/2834
2024-07-05 16:29:56.503 MSK [48084] LOG:  syncing data directory (fsync), elapsed time:
30.00 s, current path: ./base/16385/2619_vm
2024-07-05 16:30:06.533 MSK [48084] LOG:  syncing data directory (fsync), elapsed time:
40.03 s, current path: ./base/1/2615_vm
2024-07-05 16:30:10.430 MSK [48084] LOG:  database system was not properly shut down;
automatic recovery in progress
2024-07-05 16:30:10.460 MSK [48084] LOG:  invalid record length at 0/1914E18: expected at
least 24, got 0
2024-07-05 16:30:10.461 MSK [48084] LOG:  redo is not required
2024-07-05 16:30:10.514 MSK [48082] LOG:  checkpoint starting: end-of-recovery immediate
wait
2024-07-05 16:30:10.638 MSK [48082] LOG:  checkpoint complete: wrote 3 buffers (0.0%); 0
WAL file(s) added, 0 removed, 0 recycled; write=0.038 s, sync=0.018 s, total=0.143 s;
sync files=2, longest=0.010 s, average=0.009 s; distance=0 kB, estimate=0 kB;
lsn=0/1914E18, redo lsn=0/1914E18
2024-07-05 16:30:10.671 MSK [48081] LOG:  database system is ready to accept connections
```

Перед тем, как начать принимать соединения, СУБД выполнила восстановление (automatic recovery in progress).