Блокировки

Блокировки в оперативной памяти

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

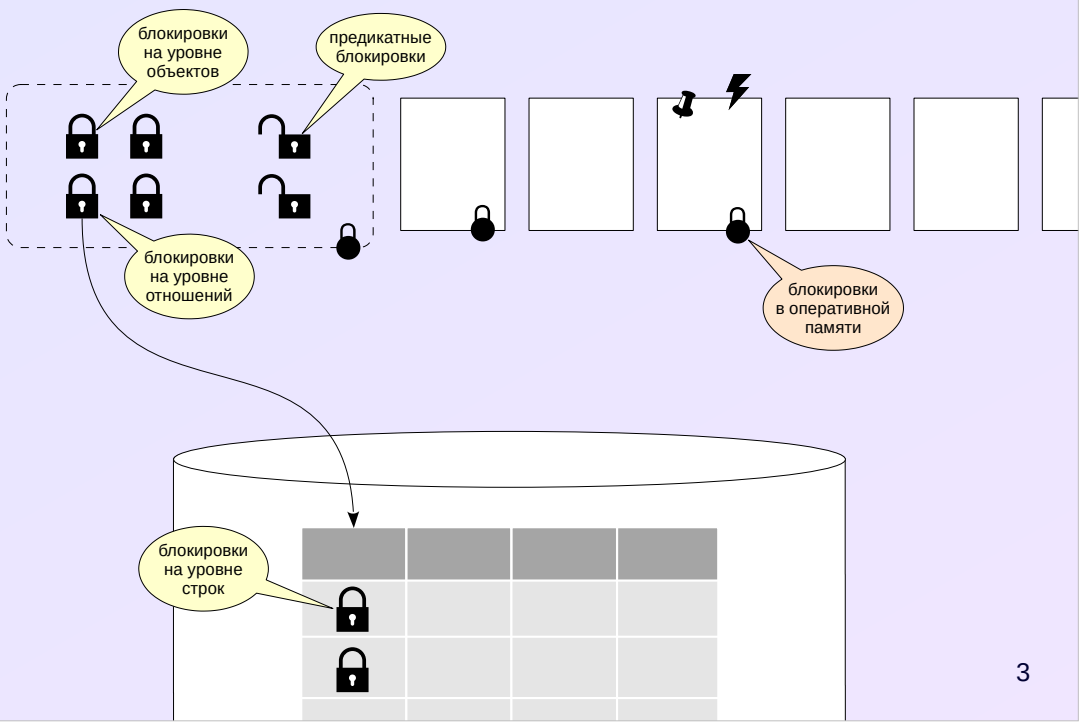
Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Блокировки в памяти

Мониторинг ожиданий

Блокировки в памяти



Спин-блокировки

- Устанавливаются на очень короткое время
  - несколько инструкций процессора
- Исключительный режим
- Нет возможности мониторинга
- Нет обнаружения взаимоблокировок
- Цикл активного ожидания
  - используются атомарные инструкции процессора

Мы уже рассмотрели «тяжелые» блокировки, действующие как правило до конца транзакции и поддерживающие множество режимов. Для защиты структур в оперативной памяти, разделяемой несколькими процессами, используются более простые (и дешевые в смысле накладных расходов) блокировки.

Самые простые из них — спин-блокировки или спинлоки (spinlock). Они предназначены для захвата на очень короткое время (несколько инструкций процессора) и защищают отдельные поля от одновременного изменения.

Спин-блокировки реализуются на основе атомарных инструкций процессора, например, compare-and-swap (https://ru.wikipedia.org/wiki/Сравнение_с_обменом).

Они захватываются только в исключительном режиме. Если блокировка занята, выполняется цикл активного ожидания — команда повторяется до тех пор, пока не выполнится успешно. Это имеет смысл, поскольку спин-блокировки применяются только в тех случаях, когда вероятность конфликта очень мала.

Спин-блокировки не обеспечивают обнаружения взаимоблокировок (за этим следят разработчики PostgreSQL) и не предоставляют никаких средств мониторинга. По большому счету, единственное, что мы можем сделать со спин-блокировками — знать о их существовании.

Легкие блокировки

- Устанавливаются на короткое время
  - обычно доли секунды
- Исключительный и разделяемый режимы
- Есть мониторинг
  - pg_stat_activity
- Нет обнаружения взаимоблокировок
- «Нечестная» очередь
  - разделяемая блокировка позволяет читающим процессам проходить без очереди

Следом идут так называемые легкие блокировки (lightweight locks, lwlocks).

Их захватывают на короткое время, которое требуется для работы со структурой данных (например, с хеш-таблицей или списком указателей). Как правило, легкая блокировка удерживается недолго, но в процессе ее удержания могут выполняться операции ввода-вывода, так что время может оказаться и значительным.

Используются два режима блокирования: исключительный (для записи) и разделяемый (для чтения).

Поддерживается очередь ожидания. Однако, пока блокировка удерживается в разделяемом режиме, другие читающие процессы проходят вне очереди. В системах с высокой степенью параллельности и большой нагрузкой это может приводить к непредсказуемо долгим ожиданиям процессов, которым требуется менять данные (https://postgrespro.ru/list/thread-id/2400193).

Проверка взаимоблокировок не выполняется, как и для спин-блокировок.

Однако легкие блокировки имеют средства для мониторинга.

Закрепление буфера

- Устанавливается на время работы с буфером
  - возможно, длительное
- Разделяемый режим
- Есть мониторинг
  - pg_stat_activity
  - pg_buffercache
- Нет обнаружения взаимоблокировок
- Пассивное ожидание
  - но обычно закрепленный буфер пропускается

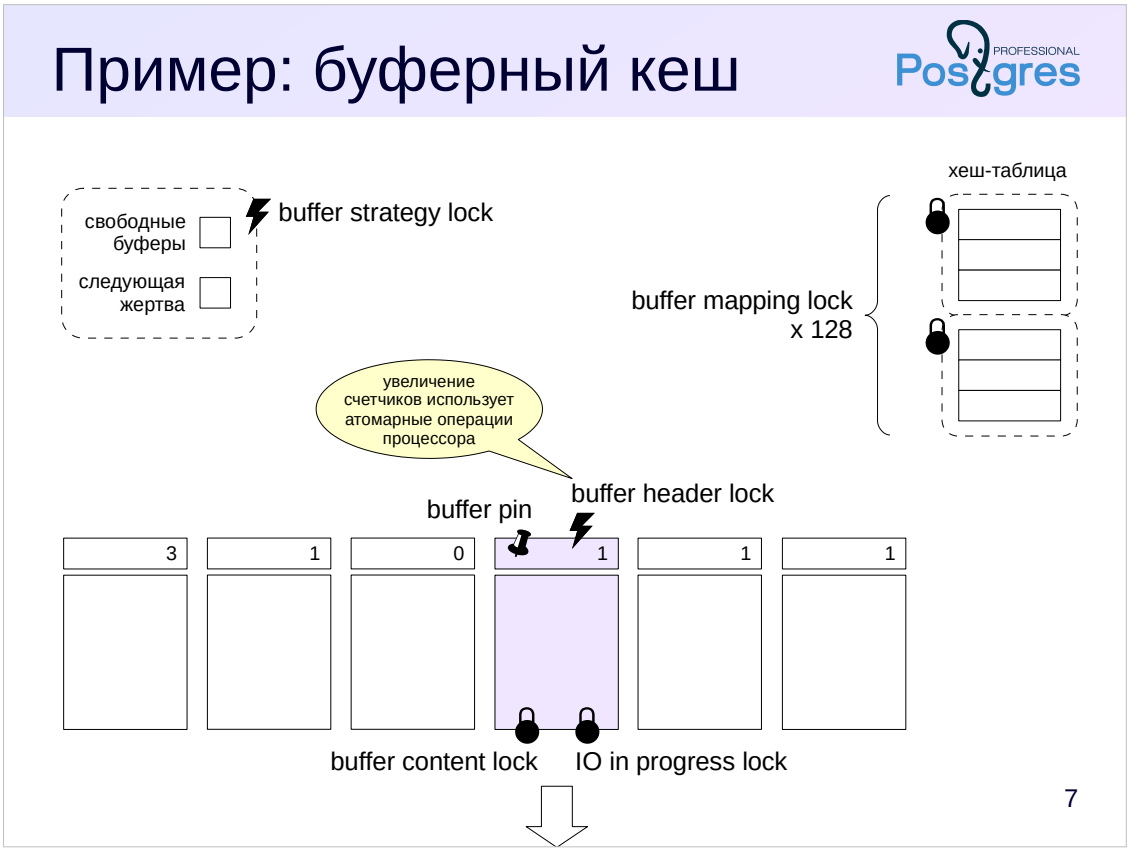
Еще один вид блокировки, который мы уже рассматривали в теме «Буферный кеш» модуля «Журналирование» — закрепление буфера (buffer pin). Для каждого буфера ведется список работающих с ним процессов.

Если буфер закреплен, в нем запрещены некоторые действия. Например, в страницу можно вставить новую строку (которую остальные процессы не увидят благодаря многоверсионности), но нельзя заменить одну страницу на другую.

Как правило, процессы пропускают закрепленный буфер и выбирают другой, чтобы не ждать снятия закрепления. Но в некоторых случаях, когда требуется именно этот буфер, запрашивается легкая блокировка в исключительном режиме. При необходимости процесс ждет, пока все остальные процессы закончат работу с буфером. Ожидание при этом пассивное: система «будит» ожидающий процесс, когда закрепление снимается.

Взаимоблокировки невозможны, за этим следят разработчики PostgreSQL.

Ожидания, связанные с закреплением, доступны для мониторинга через представление pg_stat_activity. Текущее количество закреплений буфера показывает расширение pg_buffercache.



Чтобы получить некоторое (неполное) представление о том, как и где используются блокировки, рассмотрим пример буферного кеша.

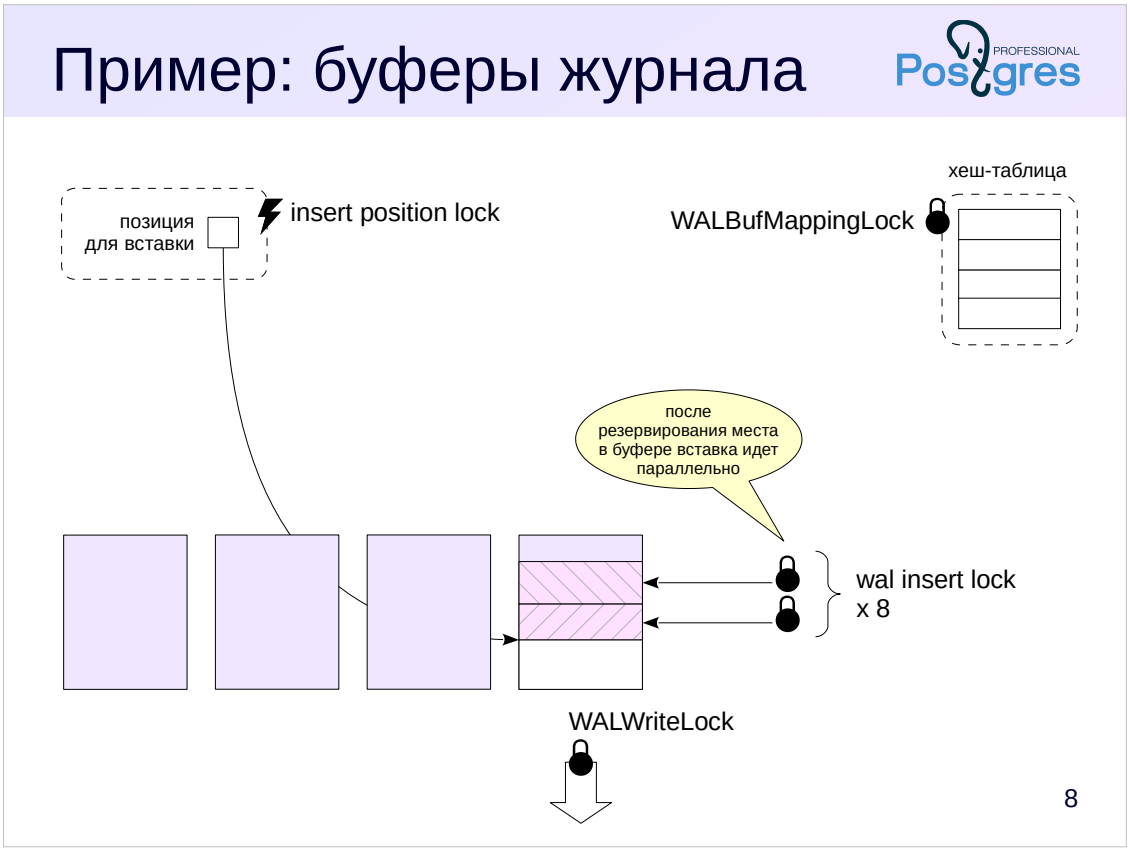
Чтобы обратиться к хеш-таблице, содержащей ссылки на буферы, процесс должен захватить легкую блокировку buffer mapping lock в разделяемом режиме, а если таблицу требуется изменять — то в исключительном режиме. Чтобы увеличить гранулярность, эта блокировка устроена как *транш*, состоящий из 128 отдельных блокировок, каждая из которых защищает свою часть хеш-таблицы.

Доступ к заголовку буфера процесс получает с помощью спин-блокировки. Отдельные операции (такие, как увеличение счетчика) могут выполняться и без явных блокировок, с помощью атомарных инструкций процессора.

Чтобы прочитать содержимое буфера, требуется блокировка buffer content lock. Обычно она захватывается только для чтения указателей на версии строк, а дальше достаточно защиты, предоставляемой закреплением буфера. Для изменения содержимого буфера эта блокировка захватывается в исключительном режиме.

При чтении буфера с диска (или записи на диск) захватывается также блокировка IO in progress, которая сигнализирует другим процессам, что страница читается — они могут встать в очередь, если им тоже нужна та же самая страница.

Указатели на свободные буферы и на следующую жертву защищены одной спин-блокировкой buffer strategy lock.



Еще один пример: буферы журнала.

Для журнального кеша тоже используется хеш-таблица, содержащая отображение страниц в буферы. В отличие от хеш-таблицы буферного кеша, эта хеш-таблица защищена единственной легкой блокировкой WALBufMappingLock, поскольку размер журнального кеша меньше (обычно 1/32 от буферного кеша) и обращение к его буферам более упорядочено.

Запись страниц на диск защищена легкой блокировкой WALWriteLock, чтобы только один процесс одновременно мог выполнять эту операцию.

Чтобы создать журнальную запись, процесс должен сначала зарезервировать место в странице. Для этого он захватывает спин-блокировку insert position lock. После того, как место зарезервировано, процесс копирует содержимое своей записи в отведенное место. Эта операция может выполняться несколькими процессами одновременно, для чего запись защищена *траншем* из 8 легких блокировок wal insert lock.

Здесь представлены не все блокировки, имеющие отношение к журналу предзаписи, но эта и предыдущая иллюстрации должны дать некоторое представление об использовании блокировок в оперативной памяти.

Мониторинг ожиданий

Ожидания в pg_stat_activity и семплирование

Типы ожиданий

Семплирование ожиданий

Когда процесс ожидает чего-либо,
этот факт отражается в представлении pg_stat_activity

- wait_event_type — тип ожидания
- wait_event — имя конкретного ожидания

Информация может быть неполна

- охвачены не все места в коде, в которых могут быть ожидания

Информация только на текущий момент

- единственный способ получить картину во времени — семплирование
- достоверная картина только при большом числе измерений

Для мониторинга ожиданий используется представление pg_stat_activity. Когда процесс (системный или обслуживающий) не может выполнять свою работу и ждет чего-либо, это ожидание можно увидеть в представлении. Столбец wait_event_type показывает тип ожидания, а столбец wait_event — имя конкретного ожидания.

Следует учитывать, что представление показывает только те ожидания, которые соответствующим образом обрабатываются в исходном коде. Если представление не показывает ожидание, это вообще говоря не означает со 100-процентной вероятностью, что процесс действительно ничего не ждет.

К сожалению, единственная доступная информация об ожиданиях — информация на текущий момент. Никакой накопленной статистики не ведется. Единственный способ получить картину ожиданий во времени — семплирование состояния представления с определенным интервалом. Встроенных средств для этого не предусмотрено, но можно использовать расширения, например, pg_wait_sampling.

https://github.com/postgrespro/pg_wait_sampling

При семплировании надо учитывать его вероятностный характер. Чтобы получить более или менее достоверную картину, число измерений должно быть достаточно высоко. Поэтому семплирование с низкой частотой не даст достоверной картины, а повышение частоты приводит к увеличению накладных расходов. По той же причине семплирование бесполезно для анализа короткоживущих сеансов.

Типы ожиданий

Блокировки

блокировки объектов	Lock
легкие блокировки	LWLock
закрепление буфера	BufferPin

Другие ожидания

ввод-вывод	IO
получение данных от другого процесса	IPC
получение данных от клиента	Client
активности в модуле расширения	Extension
«безделье»	Activity, Timeout

Все остальное — неучтенное время

Все ожидания можно разделить на несколько типов. Ожидания рассмотренных блокировок составляют большую категорию: ожидание блокировок объектов (значение Lock в столбце wait_event_type), ожидание легких блокировок (LWLock) и ожидание закрепленного буфера (BufferPin).

Но процессы могут ожидать и другие события. Ожидания ввода-вывода (IO) возникают, когда процессу требуется записать или прочитать данные. Процесс может ждать данные, необходимые для работы, от клиента (Client) или от другого процесса (IPC).

Расширения могут регистрировать свои специфические ожидания (Extension).

Бывают ситуации, когда процесс просто не выполняет полезной работы. К этой категории относится ожидание фоновых процессов в своем основном цикле (Activity), ожидание таймера (Timeout). Как правило, такие ожидания «нормальны» и не говорят о каких-либо проблемах.

Тип ожидания сопровождается именем конкретного ожидания: https://postgrespro.ru/docs/postgresql/13/monitoring-stats#WAIT-EVENT-TABLE

Если имя ожидания не определено, процесс не находится в состоянии ожидания. Такое время следует считать неучтенным, так как на самом деле неизвестно, что именно происходит в этот момент.

В следующей демонстрации мы используем файловую систему FUSE (https://github.com/libfuse/libfuse) и проект slowfs, построенный с ее помощью (https://github.com/nirs/slowfs).

Мониторинг ожиданий

Текущие ожидания можно посмотреть в представлении pg_stat_activity, которое показывает информацию о работающих процессах. Выберем только часть полей:

```
=> SELECT pid, backend_type, wait_event_type, wait_event
FROM pg_stat_activity;
```

```
  pid  |         backend_type         | wait_event_type |     wait_event
-------+------------------------------+-----------------+--------------------
 23350 | logical replication launcher | Activity        | LogicalLauncherMain
 23348 | autovacuum launcher          | Activity        | AutoVacuumMain
 23385 | client backend               |                 |
 23346 | background writer            | Activity        | BgWriterMain
 23345 | checkpointer                 | Activity        | CheckpointerMain
 23347 | walwriter                    | Activity        | WalWriterMain
(6 rows)
```

Пустые значения говорят о том, что процесс ничего не ждет и выполняет полезную работу.

Чтобы получить более или менее полную картину ожиданий процесса, требуется выполнять семплирование с некоторой частотой. Воспользуемся расширением pg_wait_sampling.

Расширение уже установлено из пакета в ОС виртуальной машины курса, но его необходимо внести в конфигурационный параметр shared_preload_libraries (что требует перезагрузки сервера).

```
=> ALTER SYSTEM SET shared_preload_libraries = 'pg_wait_sampling';
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 main restart
```

```
student$ psql
```

Теперь установим расширение в базе данных.

```
=> CREATE DATABASE locks_memory;
```

```
CREATE DATABASE
```

```
=> \c locks_memory
```

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

```
=> CREATE EXTENSION pg_wait_sampling;
```

```
CREATE EXTENSION
```

Расширение позволяет просмотреть некоторую историю ожиданий, которая хранится в кольцевом буфере. Но наиболее интересно увидеть профиль ожиданий — накопленную статистику за все время работы.

Подождем несколько секунд...

```
=> SELECT * FROM pg_wait_sampling_profile;
```

```
  pid  | event_type |         event         | queryid | count
-------+------------+-----------------------+---------+-------
 23559 | IO         | WALSync               |       0 |     2
 23559 | IO         | DataFileSync          |       0 |     5
 23562 | Activity   | AutoVacuumMain        |       0 |   698
 23559 | IO         | SLRUFlushSync         |       0 |     4
 23561 | Activity   | WalWriterMain         |       0 |   698
 23559 | IO         | ControlFileSyncUpdate |       0 |     3
 23559 | Activity   | CheckpointerMain      |       0 |   674
 23600 | IO         | CopyFileRead          |       0 |     1
 23651 | Client     | ClientRead            |       0 |   292
 23560 | Activity   | BgWriterMain          |       0 |   698
 23565 | Activity   | LogicalLauncherMain   |       0 |   698
 23651 | IO         | WALSync               |       0 |     1
 23600 | IPC        | CheckpointDone        |       0 |    24
 23600 | Client     | ClientRead            |       0 |    10
(14 rows)
```

Поскольку за прошедшее после запуска сервера время ничего не происходило, основные ожидания относятся к типу Activity (служебные процессы ждут, пока появится работа) и Client (psql ждет, пока пользователь пришлет запрос).

С установками по умолчанию частота семплирования — 100 раз в секунду. Поэтому, чтобы оценить длительность ожиданий в секундах, значение count надо делить на 100.

Чтобы понять, к какому процессу относятся ожидания, добавим к запросу представление pg_stat_activity:

```
=> SELECT p.pid, a.backend_type, a.application_name AS app, p.event_type, p.event, p.count
FROM pg_wait_sampling_profile p
  LEFT JOIN pg_stat_activity a ON p.pid = a.pid
ORDER BY p.pid, p.count DESC;
```

```
  pid  |         backend_type          | app  | event_type |         event         | count
-------+-------------------------------+------+------------+-----------------------+-------
 23559 | checkpointer                  |      | Activity   | CheckpointerMain      |   684
 23559 | checkpointer                  |      | IO         | DataFileSync          |     5
 23559 | checkpointer                  |      | IO         | SLRUFlushSync         |     4
 23559 | checkpointer                  |      | IO         | ControlFileSyncUpdate |     3
 23559 | checkpointer                  |      | IO         | WALSync               |     2
 23560 | background writer             |      | Activity   | BgWriterMain          |   708
 23561 | walwriter                     |      | Activity   | WalWriterMain         |   705
 23561 | walwriter                     |      | IO         | WALSync               |     3
 23562 | autovacuum launcher           |      | Activity   | AutoVacuumMain        |   708
 23565 | logical replication launcher  |      | Activity   | LogicalLauncherMain   |   708
 23600 |                               |      | IPC        | CheckpointDone        |    24
 23600 |                               |      | Client     | ClientRead            |    10
 23600 |                               |      | IO         | CopyFileRead          |     1
 23651 | client backend                | psql | Client     | ClientRead            |   302
 23651 | client backend                | psql | IO         | WALSync               |     1
(15 rows)
```

Дадим нагрузку с помощью pgbench и посмотрим, как изменится картина.

```
student$ pgbench -i locks_memory

dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.24 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.74 s (drop tables 0.00 s, create tables 0.03 s, client-side generate 0.34 s, vacuum 0.14 s, primary keys 0.23 s).
```

Сбрасываем собранный профиль в ноль и запускаем тест на 30 секунд в отдельном процессе. Одновременно будем смотреть, как изменяется профиль.

```
=> SELECT pg_wait_sampling_reset_profile();

 pg_wait_sampling_reset_profile
--------------------------------

(1 row)


student$ pgbench -T 30 locks_memory
```

```
=> SELECT p.pid, a.backend_type, a.application_name AS app, p.event_type, p.event, p.count
FROM pg_wait_sampling_profile p
  LEFT JOIN pg_stat_activity a ON p.pid = a.pid
WHERE a.application_name = 'pgbench'
ORDER BY p.pid, p.count DESC;

  pid  |  backend_type  |   app   | event_type |  event   | count
-------+----------------+---------+------------+----------+-------
 23974 | client backend | pgbench | IO         | WALSync  |   151
 23974 | client backend | pgbench | IO         | WALWrite |     1
(2 rows)


=> \g

  pid  |  backend_type  |   app   | event_type |   event    | count
-------+----------------+---------+------------+------------+-------
 23974 | client backend | pgbench | IO         | WALSync    |  1156
 23974 | client backend | pgbench | IO         | WALWrite   |     3
 23974 | client backend | pgbench | Client     | ClientRead |     1
(3 rows)


=> \g

  pid  |  backend_type  |   app   | event_type |   event    | count
-------+----------------+---------+------------+------------+-------
 23974 | client backend | pgbench | IO         | WALSync    |  2158
 23974 | client backend | pgbench | IO         | WALWrite   |    10
 23974 | client backend | pgbench | Client     | ClientRead |     4
(3 rows)
```

Ожидания процесса pgbench будут получаться разными в зависимости от конкретной системы. В нашем случае с большой вероятностью будет представлено ожидание записи и синхронизации журнала (IO/WALSync, IO/WALWrite).

```
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
```

```
duration: 30 s
number of transactions actually processed: 4224
latency average = 7.104 ms
tps = 140.774550 (including connections establishing)
tps = 140.830502 (excluding connections establishing)
```

Легкие блокировки

Всегда нужно помнить, что отсутствие какого-либо ожидания при семплировании не говорит о том, что ожидания не было. Если оно было короче, чем период семплирования (сотая часть секунды в нашем примере), то могло просто не попасть в выборку.

Поэтому легкие блокировки скорее всего не появились в профиле — но появятся, если собирать данные в течении длительного времени.

Чтобы гарантированно посмотреть на них, подключимся к кластеру slow с замедленной файловой системой: любая операция ввода-вывода будет занимать 1/10 секунды.

```
student$ sudo pg_ctlcluster 13 main stop
```

```
student$ sudo pg_ctlcluster 13 slow start
```

Еще раз сбросим профиль и дадим нагрузку.

```
=> \c
```

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

```
=> SELECT pg_wait_sampling_reset_profile();
```

```
 pg_wait_sampling_reset_profile
--------------------------------

(1 row)
```

```
student$ pgbench -T 30 locks_memory
```

```
=> SELECT p.pid, a.backend_type, a.application_name AS app, p.event_type, p.event, p.count
FROM pg_wait_sampling_profile p
  LEFT JOIN pg_stat_activity a ON p.pid = a.pid
WHERE a.application_name = 'pgbench'
ORDER BY p.pid, p.count DESC;
```

```
  pid  |  backend_type  |   app   | event_type |    event     | count
-------+----------------+---------+------------+--------------+-------
 24276 | client backend | pgbench | LWLock     | WALWrite     |   167
 24276 | client backend | pgbench | IO         | DataFileRead |     1
(2 rows)
```

```
=> \g
```

```
  pid  |  backend_type  |   app   | event_type |     event      | count
-------+----------------+---------+------------+----------------+-------
 24314 | client backend | pgbench | LWLock     | WALWrite       |   626
 24314 | client backend | pgbench | IO         | WALWrite       |   405
 24314 | client backend | pgbench | IO         | DataFileExtend |    20
 24314 | client backend | pgbench | IO         | WALSync        |    17
 24314 | client backend | pgbench | IO         | DataFileRead   |     2
(5 rows)
```

```
=> \g
```

```
  pid  |  backend_type  |   app   | event_type |     event      | count
-------+----------------+---------+------------+----------------+-------
 24314 | client backend | pgbench | IO         | WALWrite       |  1142
 24314 | client backend | pgbench | LWLock     | WALWrite       |   908
 24314 | client backend | pgbench | IO         | WALSync        |    26
 24314 | client backend | pgbench | IO         | DataFileExtend |    20
 24314 | client backend | pgbench | IO         | DataFileRead   |     2
(5 rows)
```

Теперь основное ожидание процесса pgbench связано с вводом-выводом, точнее с записью журнала, которая выполняется в синхронном режиме при каждой фиксации. Поскольку (вспомним слайд презентации) запись журнала на диск защищена легкой блокировкой WALWriteLock, она также присутствует в профиле.

```
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 17
latency average = 1788.184 ms
tps = 0.559227 (including connections establishing)
tps = 0.561565 (excluding connections establishing)
```

Итоги

Блокировки в оперативной памяти реализуются по-разному

- спин-блокировки, легкие блокировки, закрепление буфера
- относительно короткое время и облегченная инфраструктура

Мониторинг текущих ожиданий с помощью представления pg_stat_activity

- семплирование для получения картины во времени

Практика

1. Открытый курсор удерживает закрепление буфера, чтобы чтение следующей строки выполнялось быстрее. Убедитесь в этом с помощью расширения pg_buffercache.
2. Откройте курсор по таблице и, не закрывая его, выполните очистку таблицы (VACUUM). Будет ли очистка ждать освобождения закрепления буфера?
3. Повторите эксперимент, выполнив очистку с заморозкой (VACUUM FREEZE). Убедитесь, что в профиль ожиданий обслуживающего процесса попало ожидание закрепления буфера.

1. Расширение pg_buffercache было рассмотрено в модуле «Журнал», тема «Буферный кеш».

https://www.postgrespro.ru/docs/postgresql/13/pgbuffercache.html

Представление pg_buffercache содержит столбец pinning_backends, который показывает количество процессов, закрепивших этот буфер. Нужный буфер можно найти по условию relfilenode = pg_relation_filenode(*имя_таблицы*).

2. Для проверки удобно воспользоваться вариантом команды очистки VACUUM VERBOSE.

1. Закрепление буфера при открытом курсоре

Сначала выполняем подготовительные действия.

Устанавливаем необходимые расширения:

```
=> ALTER SYSTEM SET shared_preload_libraries = 'pg_wait_sampling';

ALTER SYSTEM

student$ sudo pg_ctlcluster 13 main restart

student$ psql

=> CREATE DATABASE locks_memory;

CREATE DATABASE

=> \c locks_memory

You are now connected to database "locks_memory" as user "student".

=> CREATE EXTENSION pg_wait_sampling;

CREATE EXTENSION

=> CREATE EXTENSION pg_buffercache;

CREATE EXTENSION
```

Таблица, как в предыдущих практиках:

```
=> CREATE TABLE accounts(acc_no integer, amount numeric);

CREATE TABLE

=> INSERT INTO accounts VALUES (1,1000.00),(2,2000.00),(3,3000.00);

INSERT 0 3
```

Начинаем транзакцию, открываем курсор и выбираем одну строку.

```
=> BEGIN;

BEGIN

=> DECLARE c CURSOR FOR SELECT * FROM accounts;

DECLARE CURSOR

=> FETCH c;

 acc_no | amount
--------+---------
      1 | 1000.00
(1 row)
```

Проверим, закреплен ли буфер:

```
=> SELECT * FROM pg_buffercache
WHERE relfilenode = pg_relation_filenode('accounts') AND relforknumber = 0 \gx

-[ RECORD 1 ]----+------
bufferid         | 400
relfilenode      | 33609
reltablespace    | 1663
reldatabase      | 33585
relforknumber    | 0
relblocknumber   | 0
isdirty          | t
usagecount       | 4
pinning_backends | 1
```

Да, pinning_backends = 1.

2. Очистка закрепленного буфера

Выполним очистку:

```
student$ psql locks_memory
```

```
=> VACUUM VERBOSE accounts;
```

```
INFO:  vacuuming "public.accounts"
INFO:  "accounts": found 0 removable, 0 nonremovable row versions in 1 out of 1 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 159554
There were 0 unused item identifiers.
Skipped 1 page due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
INFO:  vacuuming "pg_toast.pg_toast_33609"
INFO:  "pg_toast_33609": found 0 removable, 0 nonremovable row versions in 0 out of 0 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 159554
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
```

Как мы видим, страница была пропущена: Skipped 1 page due to buffer pins.

Очистка не может обработать страницу: если буфер закреплен, из страницы запрещено удалять версии строк. Но очистка не ждет, пока буфер освободится — страница будет обработана при следующей очистке.

3. Заморозка закрепленного буфера

Выполняем очистку с заморозкой:

```
=> VACUUM FREEZE VERBOSE accounts;
```

Очистка зависает до закрытия курсора. При явно запрошенной заморозке нельзя пропустить ни одну страницу, не отмеченную в карте заморозки — иначе невозможно уменьшить максимальный возраст незамороженных транзакций в pg_class.relfrozenxid.

```
=> SELECT age(relfrozenxid) FROM pg_class WHERE oid = 'accounts'::regclass;
```

```
 age
-----
   2
(1 row)
```

```
=> COMMIT;
```

```
COMMIT
```

```
INFO:  aggressively vacuuming "public.accounts"
INFO:  "accounts": found 0 removable, 3 nonremovable row versions in 1 out of 1 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 159554
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 5.18 s.
INFO:  aggressively vacuuming "pg_toast.pg_toast_33609"
INFO:  "pg_toast_33609": found 0 removable, 0 nonremovable row versions in 0 out of 0 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 159554
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
```

```
=> SELECT age(relfrozenxid) FROM pg_class WHERE oid = 'accounts'::regclass;
```

```
 age
-----
   0
(1 row)
```

Профиль ожиданий:

```
=> SELECT p.pid, a.backend_type, a.application_name AS app, p.event_type, p.event, p.count
FROM pg_wait_sampling_profile p
  LEFT JOIN pg_stat_activity a ON p.pid = a.pid
ORDER BY p.pid, p.count DESC;
```

```
  pid  |         backend_type          | app  | event_type |         event          | count
-------+-------------------------------+------+------------+------------------------+-------
 48694 | checkpointer                  |      | Activity   | CheckpointerMain       |   900
 48694 | checkpointer                  |      | IO         | SLRUFlushSync          |     4
 48694 | checkpointer                  |      | IO         | DataFileSync           |     3
 48694 | checkpointer                  |      | IO         | ControlFileSyncUpdate  |     2
 48694 | checkpointer                  |      | IO         | WALSync                |     2
 48694 | checkpointer                  |      | IO         | WALWrite               |     1
 48695 | background writer             |      | Activity   | BgWriterMain           |   919
 48696 | walwriter                     |      | Activity   | WalWriterMain          |   915
 48696 | walwriter                     |      | IO         | WALSync                |     4
 48697 | autovacuum launcher           |      | Activity   | AutoVacuumMain         |   919
 48700 | logical replication launcher  |      | Activity   | LogicalLauncherMain    |   919
 48735 |                               |      | IPC        | CheckpointDone         |    18
 48735 |                               |      | Client     | ClientRead             |     4
 48735 |                               |      | IO         | WALSync                |     1
 48783 | client backend                | psql | Client     | ClientRead             |   535
 48783 | client backend                | psql | IO         | WALSync                |     4
 48783 | client backend                | psql | IO         | DataFileImmediateSync  |     1
 49059 | client backend                | psql | BufferPin  | BufferPin              |   485
 49059 | client backend                | psql | Client     | ClientRead             |    25
(19 rows)
```

Тип ожидания BufferPin говорит о том, что очистка ждала освобождения буфера.