Operating Systems

Caching

Me

April 12, 2016

План лекции

- Свободная память - бесполезная память.
- Медленный доступ к диску и Buffer Cache.
- Доступ к файлам и Page Cache.

Потребление памяти процессами

- Большинство процессов в системе потребляет небольшое количество ресурсов
  - например, у меня в системе в среднем около 250 процессов;
  - все вместе они используют около 3 GB памяти из 8 GB;
  - больше половины занятой памяти используется 10 % процессов;

Потребление памяти процессами

- Большинство процессов в системе потребляет небольшое количество ресурсов
  - например, у меня в системе в среднем около 250 процессов;
  - все вместе они используют около 3 GB памяти из 8 GB;
  - больше половины занятой памяти используется 10 % процессов;
- Свободная память не приносит пользы:
  - без разницы 100 GB памяти или 200 GB у вас в системе, если вы используете только 3 GB;
  - полезно держать небольшой запас память на всякий случай - но нам не нужно для этого несколько GB

Buffer Cache

- Доступ к диску медленный:
  - доступ к диску как правило обладает локальностью:
    - временная локальность - обращение к данным, к которым мы уже обращались недавно;
    - пространственная локальность - обращение к данным, которые находятся на диске рядом с данными, к которым мы обращались;

Buffer Cache

- Доступ к диску медленный:
  - доступ к диску как правило обладает локальностью:
    - временная локальность - обращение к данным, к которым мы уже обращались недавно;
    - пространственная локальность - обращение к данным, которые находятся на диске рядом с данными, к которым мы обращались;
- У нас есть запас в несколько GB свободной памяти:
  - кеш в несколько GB при наличии локальности может дать большой прирост в производительности;
  - мы всегда можем уменьшить кеш, если нам не хватает памяти;

Стратегии замещения и записи

- Размер кеша ограничен
  - когда-нибудь он заполнится и придется решать кого выкинуть из кеша;
  - выкинем сектор, к которому дольше всего не было обращений - LRU;

Стратегии замещения и записи

- Размер кеша ограничен
  - когда-нибудь он заполнится и придется решать кого выкинуть из кеша;
  - выкинем сектор, к которому дольше всего не было обращений - LRU;
- Buffer Cache позволяет ускорять запись
  - мы записываем данные в кеш - дальше кеш разберется сам;
  - нужно решить когда записывать данные на диск
    - мы можем начать запись на диск, как только мы добавили данные в кеш - write-through;
    - мы можем отложить запись на какое-то время - write-back;

Page Cache

- Пользователь, обычно, не обращается к диску - он обращается к файлу
  - где и как этот файл располагается на диске не особо важно;
  - ФС использует Buffer Cache для ускорения работы с диском;

Page Cache

- Пользователь, обычно, не обращается к диску - он обращается к файлу
  - где и как этот файл располагается на диске не особо важно;
  - ФС использует Buffer Cache для ускорения работы с диском;
- Зачем просить ФС делать лишнюю работу и искать нужный сектор диска?
  - давайте кешировать доступ к файлам вместо доступа к диску;
  - в Linux Kernel это называется Page Cache (судя по всему название пришло из SVR4);

Page Cache в Linux Kernel

```
1  struct address_space {
2    struct inode                                *host;
3    struct radix_tree_root                      page_tree;
4
5    spinlock_t                                  tree_lock;
6    atomic_t                                    i_mmap_writable;
7    struct rb_root                              i_mmap;
8    struct rw_semaphore                         i_mmap_rwsem;
9    unsigned long                               nrpages;
10   unsigned long                               nrexceptional;
11   pgoff_t                                     writeback_index;
12
13   const struct address_space_operations *a_ops;
14
15   unsigned long                               flags;
16   spinlock_t                                  private_lock;
17   struct list_head                            private_list;
18   void                                        *private_data;
19 };
```

Page Cache в Linux Kernel

```
1  struct address_space_operations {
2    int (*writepage)(struct page *page, struct writeback_control *wbc);
3    int (*readpage)(struct file *, struct page *);
4    int (*writepages)(struct address_space *, struct writeback_control *);
5
6    int (*set_page_dirty)(struct page *page);
7
8    int (*readpages)(struct file *filp, struct address_space *mapping,
9                     struct list_head *pages, unsigned nr_pages);
10
11   int (*write_begin)(struct file *, struct address_space *mapping,
12                      loff_t pos, unsigned len, unsigned flags,
13                      struct page **pagep, void **fsdata);
14   int (*write_end)(struct file *, struct address_space *mapping,
15                    loff_t pos, unsigned len, unsigned copied,
16                    struct page *page, void *fsdata);
17
18   ...
19 };
```

Откуда берутся address_space_operations?

- Их выставляет ФС при открытии файла:
  - ядро алоцирует структуру *struct inode*;
  - *struct address_space* - поле этой структуры;
  - ФС заполняет *struct inode* при открытии файла;

Чтение файла

```
1  static ssize_t do_generic_file_read(struct file *filp, loff_t *ppos,
2                                      struct iov_iter *iter, ...)
3  {
4     ...
5     index = *ppos >> PAGE_CACHE_SHIFT;
6     ...
7     page = find_get_page(mapping, index);
8     ...
9     error = mapping->a_ops->readpage(filp, page);
10    ...
11    ret = copy_page_to_iter(page, offset, nr, iter);
12    ...
13 }
```

Запись файла

- В отличие от чтения, запись в файл более сложный процесс
  - запись состоит из двух частей: запись в кеш и сброс кеша на диск;
- *write_begin*/*write_end* и *set_page_dirty* используются в первой части - при записи в кеш;
- *writepage*/*writepages* используются для записи страниц кеша на диск;

Запись файла

```
1  ssize_t generic_perform_write(struct file *file,
2                                struct iov_iter *i, loff_t pos)
3  {
4     ...
5     status = a_ops->write_begin(file, mapping, pos, bytes, flags,
6                                 &page, &fsdata);
7     ...
8     copied = iov_iter_copy_from_user_atomic(page, i, offset, bytes);
9     ...
10    status = a_ops->write_end(file, mapping, pos, bytes, copied,
11                              page, fsdata);
12    ...
13 }
```

- *write_begin* занимается подготовкой к записи:
  - ищет/аллоцирует страницы в Page Cache;
  - читает данные - запрос на запись может быть не выровнен;
- *write_end* освобождает ресурсы и обновляет метаданные;

Запись файла

- Оставшаяся часть - запись страниц файла на диск:
  - никакого секрета или хитрого трюка;
  - Linux Kernel честно поддерживает список "грязных" inode;
  - для каждого inode Page Cache отслеживает "грязные" страницы;
  - для записи используется *writepage* из *address_space_operations*;

Запись файла

- Оставшаяся часть - запись страниц файла на диск:
  - никакого секрета или хитрого трюка;
  - Linux Kernel честно поддерживает список "грязных" inode;
  - для каждого inode Page Cache отслеживает "грязные" страницы;
  - для записи используется *writepage* из *address_space_operations*;
- Запись страниц происходит:
  - по требованию - sync;
  - по необходимости - мало памяти или много "грязных" страниц;
  - периодически - мы не хотим, чтобы данные долго оставались "грязными" в кеше;

Последствия кеширования

- Важно закрывать файлы:
  - это позволит сбросить буфер библиотеки, которую вы используете;
  - это позволит освободить файловый дескриптор - их количество ограничено;
  - это не гарантирует, что файлы записались на диск;

Последствия кеширования

- Важно закрывать файлы:
  - это позволит сбросить буфер библиотеки, которую вы используете;
  - это позволит освободить файловый дескриптор - их количество ограничено;
  - это не гарантирует, что файлы записались на диск;
- Как гарантировать, что все закешированные данные *файла* реально записались на диск?
  - для этого есть два вызова *fsync*/*fdatasync*;
  - если вы только создали файл, то даже это не поможет;
  - вам нужно также сделать *fsync* на родительском каталоге;

Отображение в память

- Page Cache находится в ядре и не доступен в userspace
  - нам приходится копировать данные из/в userspace из Page Cache;
  - такое копирование приводит к дублированию данных;
  - портит процессорный кеш;

Отображение в память

- Page Cache находится в ядре и не доступен в userspace
  - нам приходится копировать данные из/в userspace из Page Cache;
  - такое копирование приводит к дублированию данных;
  - портит процессорный кеш;
- Можем ли мы предоставить доступ к Page Cache из userspace?
  - на чтение - легко;
  - для записи нужно уметь отслеживать запись в страницу из userspace;

Page Fault Again

- Для отображения файла в память существует вызов *mmap*
  - параметры: файловый дескриптор, смещение и размер;
  - на выходе получаем указатель на обычную память;
  - *mmap* отображает часть виртуальной памяти процесса на страницы в Page Cache;

Page Fault Again

- Для отображения файла в память существует вызов *mmap*
  - параметры: файловый дескриптор, смещение и размер;
  - на выходе получаем указатель на обычную память;
  - *mmap* отображает часть виртуальной памяти процесса на страницы в Page Cache;
- Для отслеживания записей отображение должно быть Read Only
  - в обработчике Page Fault разрешаем запись и помечаем страницу в Page Cache как грязную;
  - после записи страницы делаем ее Read Only опять;