zhuanlan.zhihu.com /p/374626979

Linux 文件 I/O 进化史（一）：Buffered I/O

大纲

想找时间复习和总结一下 Linux 的文件 I/O 方式。大概想了，主要内容可以分成 4 个部分：

1. Buffered I/O：传统的基于 page cache 的文件读写。
2. mmap：可以让应用像访问内存一样访问文件。
3. Direct I/O 和 AIO：绕过 page cache 的 I/O 方式，同时支持异步文件 I/O。
4. io_uring：Linux 5.1 才引进的全新异步 I/O 方式。

本文是 **Linux 文件 I/O 进化史**系列的第一篇（后面的还没写，太久没发文章，先发出来充充数……）。

Buffered I/O

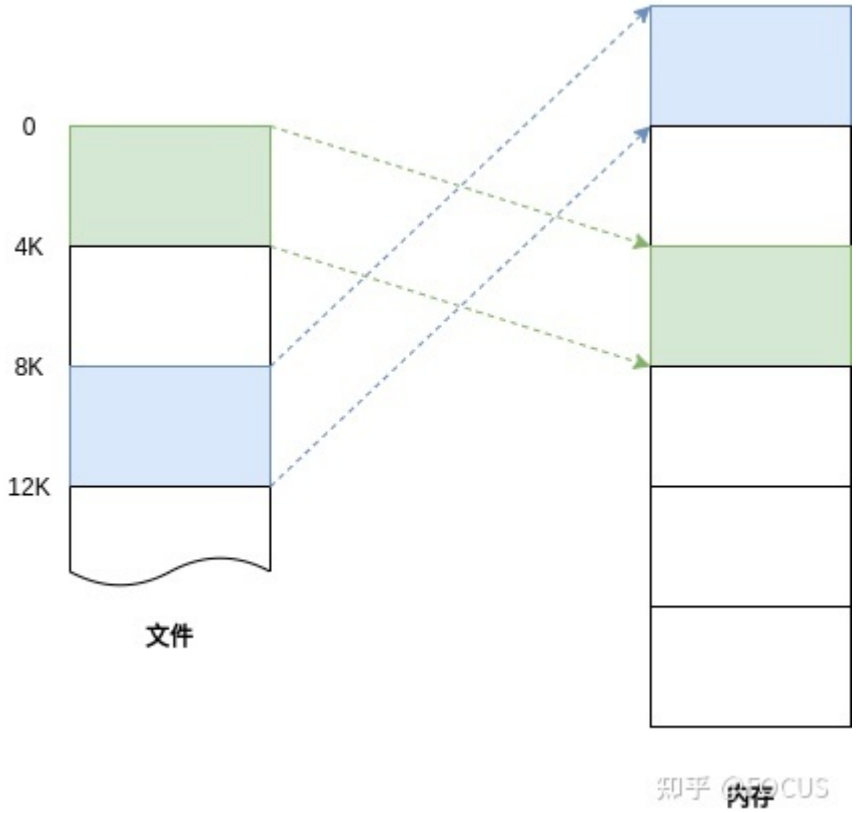
最常见的文件读写方式是直接调用 `read`/ `write` 系统调用，包括后来拓展的新接口 `pread`/ `pwrite`、`readv`/ `writev`、 `preadv`/ `pwritev`。具体接口说明可以参考 Linux manual page，这里就不多说了。

默认情况下，用这些接口读写文件的时候都会经过内核维护的 page cache，所以称之为 buffered I/O。

下图展示了 `read`/ `write` 的基本流程。读和写都可以分两种情况：

1. 命中 page cache：要读/写的数据刚好在 page cache 中，直接读/写 page cache 即可。
2. 不命中 page cache：要读/写的数据不在 page cache 中，需要到块设备上读取。一般情况下，块设备的速度比较慢，此时线程会被阻塞，发生上下文切换。

Page Cache



（图片来自 Scylla）

文件到内存之间的映射是按 page 为单位进行管理的。Buffered I/O 读写的基本单元是 page（4KB）。

如果读请求无法命中 page cache，则至少（不考虑预读）会从块设备读取 4KB 大小的数据。对于读取小数据（几字节～几百字节）的场景，如果数据的访问具有局部性，则大概率可以加快下一次读取。但是如果数据的访问完全随机，多读的这部分数据可能就浪费了，最终导致几十甚至几百倍的读放大。

如果写请求不命中 page cache，并且写入的 offset 或数据大小不与 4KB 对齐，则可能产生读 I/O —— 需要先读取相关的 page 再更新其中一部分。

下面写了一个例子测试写请求不命中 page cache 时的读 I/O 情况（测试代码 1）：

1. **当写请求不命中 page cache，且写入的 offset 和数据大小都与 4KB 对齐时，不会产生读 I/O（read_bytes 为 0）**：

```
$ ./pagecache 4096
           Files: 1
     Directories: 0
   Evicted Pages: 32768 (128M)
         Elapsed: 0.28491 seconds
file_size 134217728 write_unit 4096 write_bytes 134217728 read_bytes 0

$ ./pagecache 8192
           Files: 1
     Directories: 0
   Evicted Pages: 32768 (128M)
```

```
         Elapsed: 0.30908 seconds
file_size 134217728 write_unit 8192 write_bytes 134217728 read_bytes 0
```

1. **当写请求不命中 page cache，且写入的 offset 或数据大小不与 4KB 对齐时，会产生读 I/O（read_bytes 不为 0）：**

```
$ ./pagecache 1024
           Files: 1
     Directories: 0
   Evicted Pages: 32768 (128M)
         Elapsed: 0.31205 seconds
file_size 134217728 write_unit 1024 write_bytes 134217728 read_bytes
134217728

$ ./pagecache 5024
           Files: 1
     Directories: 0
   Evicted Pages: 32768 (128M)
         Elapsed: 0.30479 seconds
file_size 134217728 write_unit 5024 write_bytes 134217728 read_bytes
108572672
```

预读

预读，read ahead，意思就是提前读入。

对于数据的访问具有较好的顺序特性的应用，预读可以提升整体性能。但是对于数据的访问完全随机的应用，预读其实是个“性能杀手”。

下面写了个简单的程序测试下内核预读的情况：

1. **顺序读：每次读 4KB，读 64 次**（测试代码 2）：

第 1 次 read 的时候，内核预读了 3 个 page（共 4 个 page）。

第 2 次 read 的时候，内核预读了 8 个 page。

第 3、4 次 read 没产生预读。

第 5 次 read，内核预读了 16 个 page。

...

第 13 次 read，内核预读了 32 个 page。

之后的所有预读最大是 32 个 page。这个预读的最大值和 `/sys/block/<bdev>/queue/read_ahead_kb` 有关，默认是 128KB（32 个 page）。

```
$ ./readahead 4096
           Files: 1
     Directories: 0
   Evicted Pages: 32768 (128M)
         Elapsed: 0.3029 seconds
0 app read bytes 4096, os read bytes 16384
1 app read bytes 4096, os read bytes 32768
2 app read bytes 4096, os read bytes 0
3 app read bytes 4096, os read bytes 0
4 app read bytes 4096, os read bytes 65536
5 app read bytes 4096, os read bytes 0
6 app read bytes 4096, os read bytes 0
7 app read bytes 4096, os read bytes 0
8 app read bytes 4096, os read bytes 0
9 app read bytes 4096, os read bytes 0
10 app read bytes 4096, os read bytes 0
11 app read bytes 4096, os read bytes 0
12 app read bytes 4096, os read bytes 131072
13 app read bytes 4096, os read bytes 0
14 app read bytes 4096, os read bytes 0
15 app read bytes 4096, os read bytes 0
16 app read bytes 4096, os read bytes 0
17 app read bytes 4096, os read bytes 0
18 app read bytes 4096, os read bytes 0
19 app read bytes 4096, os read bytes 0
20 app read bytes 4096, os read bytes 0
21 app read bytes 4096, os read bytes 0
22 app read bytes 4096, os read bytes 0
23 app read bytes 4096, os read bytes 0
24 app read bytes 4096, os read bytes 0
25 app read bytes 4096, os read bytes 0
26 app read bytes 4096, os read bytes 0
27 app read bytes 4096, os read bytes 0
28 app read bytes 4096, os read bytes 131072
29 app read bytes 4096, os read bytes 0
30 app read bytes 4096, os read bytes 0
31 app read bytes 4096, os read bytes 0
32 app read bytes 4096, os read bytes 0
33 app read bytes 4096, os read bytes 0
34 app read bytes 4096, os read bytes 0
35 app read bytes 4096, os read bytes 0
36 app read bytes 4096, os read bytes 0
37 app read bytes 4096, os read bytes 0
38 app read bytes 4096, os read bytes 0
```

```
39 app read bytes 4096, os read bytes 0
40 app read bytes 4096, os read bytes 0
41 app read bytes 4096, os read bytes 0
42 app read bytes 4096, os read bytes 0
43 app read bytes 4096, os read bytes 0
44 app read bytes 4096, os read bytes 0
45 app read bytes 4096, os read bytes 0
46 app read bytes 4096, os read bytes 0
47 app read bytes 4096, os read bytes 0
48 app read bytes 4096, os read bytes 0
49 app read bytes 4096, os read bytes 0
50 app read bytes 4096, os read bytes 0
51 app read bytes 4096, os read bytes 0
52 app read bytes 4096, os read bytes 0
53 app read bytes 4096, os read bytes 0
54 app read bytes 4096, os read bytes 0
55 app read bytes 4096, os read bytes 0
56 app read bytes 4096, os read bytes 0
57 app read bytes 4096, os read bytes 0
58 app read bytes 4096, os read bytes 0
59 app read bytes 4096, os read bytes 0
60 app read bytes 4096, os read bytes 131072
61 app read bytes 4096, os read bytes 0
62 app read bytes 4096, os read bytes 0
63 app read bytes 4096, os read bytes 0
```

1. **随机读：每次读 4KB，随机选 offset**（测试代码 3）：**所有读操作都没有发生预读**（看样子，内核的预读算法还是比较智能的……）。

```
$ ./readahead_rand 4096
           Files: 1
     Directories: 0
   Evicted Pages: 32768 (128M)
         Elapsed: 0.28412 seconds
0 app read bytes 4096, os read bytes 4096
1 app read bytes 4096, os read bytes 4096
2 app read bytes 4096, os read bytes 4096
3 app read bytes 4096, os read bytes 4096
4 app read bytes 4096, os read bytes 4096
5 app read bytes 4096, os read bytes 4096
6 app read bytes 4096, os read bytes 4096
7 app read bytes 4096, os read bytes 4096
8 app read bytes 4096, os read bytes 4096
9 app read bytes 4096, os read bytes 4096
10 app read bytes 4096, os read bytes 4096
```

```
11 app read bytes 4096, os read bytes 4096
12 app read bytes 4096, os read bytes 4096
13 app read bytes 4096, os read bytes 4096
14 app read bytes 4096, os read bytes 4096
15 app read bytes 4096, os read bytes 4096
16 app read bytes 4096, os read bytes 4096
17 app read bytes 4096, os read bytes 4096
18 app read bytes 4096, os read bytes 4096
19 app read bytes 4096, os read bytes 4096
20 app read bytes 4096, os read bytes 4096
21 app read bytes 4096, os read bytes 4096
22 app read bytes 4096, os read bytes 4096
23 app read bytes 4096, os read bytes 4096
24 app read bytes 4096, os read bytes 4096
25 app read bytes 4096, os read bytes 4096
26 app read bytes 4096, os read bytes 4096
27 app read bytes 4096, os read bytes 4096
28 app read bytes 4096, os read bytes 4096
29 app read bytes 4096, os read bytes 4096
30 app read bytes 4096, os read bytes 4096
31 app read bytes 4096, os read bytes 4096
32 app read bytes 4096, os read bytes 4096
33 app read bytes 4096, os read bytes 4096
34 app read bytes 4096, os read bytes 4096
35 app read bytes 4096, os read bytes 4096
36 app read bytes 4096, os read bytes 4096
37 app read bytes 4096, os read bytes 4096
38 app read bytes 4096, os read bytes 4096
39 app read bytes 4096, os read bytes 4096
40 app read bytes 4096, os read bytes 4096
41 app read bytes 4096, os read bytes 4096
42 app read bytes 4096, os read bytes 4096
43 app read bytes 4096, os read bytes 4096
44 app read bytes 4096, os read bytes 4096
45 app read bytes 4096, os read bytes 4096
46 app read bytes 4096, os read bytes 4096
47 app read bytes 4096, os read bytes 4096
48 app read bytes 4096, os read bytes 4096
49 app read bytes 4096, os read bytes 4096
50 app read bytes 4096, os read bytes 4096
51 app read bytes 4096, os read bytes 4096
52 app read bytes 4096, os read bytes 4096
53 app read bytes 4096, os read bytes 4096
54 app read bytes 4096, os read bytes 4096
```

```
55 app read bytes 4096, os read bytes 4096
56 app read bytes 4096, os read bytes 4096
57 app read bytes 4096, os read bytes 4096
58 app read bytes 4096, os read bytes 4096
59 app read bytes 4096, os read bytes 4096
60 app read bytes 4096, os read bytes 4096
61 app read bytes 4096, os read bytes 4096
62 app read bytes 4096, os read bytes 4096
63 app read bytes 4096, os read bytes 4096
```

posix_fadvise

`posix_fadvise` 是一个用于控制 page cache 预读或清理策略的接口。应用可以使用这个接口来告诉内核，接下来将以何种模式访问文件数据，从而允许内核执行适当的优化。但是，这个接口对内核提交的是**建议（advise）**，不一定会被采纳。

```
int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

Advice 的取值有：

1. `POSIX_FADV_NORMAL` ：无特别建议，重置预读大小为默认值。
2. `POSIX_FADV_SEQUENTIAL` ：将要进行顺序操作，设置最大预读大小为默认值的 2 倍。
3. `POSIX_FADV_RANDOM` ：将要进行随机操作，禁用预读。
4. `POSIX_FADV_NOREUSE` ：指定的数据将只访问一次，内核无操作（no-op）。
5. `POSIX_FADV_WILLNEED` ：指定的数据即将被访问，预读数据到 page cache。
6. `POSIX_FADV_DONTNEED` ：指定的数据将不会被访问，丢弃 page cache 中的数据。

对于 `POSIX_FADV_NORMAL`、 `POSIX_FADV_RANDOM` 和 `POSIX_FADV_SEQUENTIAL` 这三个建议，内核会对文件的预读窗口大小做调整（设置为默认值、0 和默认值的 2 倍）。这些建议的影响范围是整个 fd（无视offset 和 len 参数）。

对于 `POSIX_FADV_WILLNEED` 和 `POSIX_FADV_DONTNEED` ，内核会尝试直接对 page cache 做调整。内核会根据情况采纳建议，但不会强制换入或换出。

当建议为 `POSIX_FADV_WILLNEED` 时，内核会将数据页加载到 page cache。这里根据内存负载的情况，内核可能会减少读取的数据量。

当建议为 `POSIX_FADV_DONTNEED` 时，内核先将脏页异步刷盘，并且只会尽力而为，清除掉自己能清除的缓存，而不会等待刷脏完成后再清除文件的全部缓存。

参考文档

1. Linux Performance and Tuning Tricks
2. Different I/O Access Methods for Linux, What We Chose for Scylla, and Why
3. Linux内核的文件预读

4. 测试代码 1：https://github.com/JinheLin/pagecache_test/blob/main/pagecache_write.cc
5. 测试代码 2：https://github.com/JinheLin/pagecache_test/blob/main/readahead_sequential.cc
6. 测试代码 3：https://github.com/JinheLin/pagecache_test/blob/main/readahead_random.cc