File I/O Buffering

Trong bài trước, chúng ta đã được giới thiệu 2 system call read() và write() để đọc và ghi file vào một vùng nhớ trên tầng user đến một file của ổ đĩa. Trong bài học này, chúng ta sẽ đi sâu hơn vào cách triển khai các sysem call này trong hệ thống Linux, từ đó dẫn đến yêu cầu của việc đồng bộ file trong hệ thống Linux.

Kernel buffer của file I/O

Hiểu một cách tự nhiên, chúng ta sẽ hình dung là read() đọc trực tiếp nội dung từ file trên ổ đĩa và trả nội dung vào một buffer cho chương trình gọi trên tầng user, hoặc write() ghi nội dung của một buffer trên tầng user vào một file trên ổ đĩa. Tuy nhiên trong thực tế, triển khai theo cách này có thể làm giảm hiệu năng của hệ thống vì các nguyên nhân sau:

- Thời gian xử lý của hàm read/write kéo dài do phải chờ dữ liệu được đọc/ghi vào ổ cứng, mà chúng ta đều biết là tốc độc truy xuất ổ cứng rất chậm so với tốc độ xử lý của CPU
- Giả sử chương trình của bạn gọi read/write 100 lần mà mỗi lần chỉ đọc hoặc ghi 1 byte dữ liệu vào file thì hệ thống vẫn phải truy cập vào ổ cứng 100 lần. Việc này rõ ràng là rất tốn kém, đặc biết là với nhiều thiết bị embedded bị hạn chế về phần cứng.

Cách giải quyết vấn đề này như sau: Khi chương trình gọi write(), kernel thay vì ghi trực tiếp dữ liệu vào file nằm trong ổ cứng sẽ copy vào một buffer trong kernel (kernel *buffer cache*). Các buffer đã được ghi dữ liệu này được gọi là *dirty buffer.* Sau đó, một kernel thread chạy background sẽ tự động thu thập và ghi dirty buffer vào ổ cứng, rồi "làm sạch" các dirty buffer đó để tái sử dụng. Xét một ví dụ ghi 8 byte từ user space vào file fd dưới đây:

```
write(fd, “vimentor”, 8);
```

Với dòng lệnh này, kernel chỉ kiểm tra tính hợp lệ của các đối số truyền vào, ghi chuỗi "vimentor" vào kernel buffer rồi return cho chương trình luôn. Nếu có một tiến trình nào muốn đọc những bytes này của file mà dữ liệu đó vẫn ở kernel buffer, kernel sẽ lấy các bytes dữ liệu đó từ kernel buffer đó trả về cho chương trình, thay vì truy cập vào file trong ổ cứng.

Tương tự như vậy với read(), kernel đọc dữ liệu từ ổ cứng và lưu trữ nó vào kernel buffer, các chương trình tầng user space sẽ lấy dữ liệu từ kernel buffer này. Nếu nhiều chương trình đọc file trong thời điểm kernel buffer vẫn lưu nội dung của file thì kernel cũng không cần phải đọc từ ổ cứng nữa. Việc sử dụng kernel buffer này làm cho read() và write() nhanh hơn rất rất nhiều do thời gian chết lớn nhất nằm ở bước truy suất ổ cứng. Linux kernel không quy định độ lớn của buffer cache. Kernel sẽ muốn cấp phát buffer cache mỗi khi được yêu cầu, giới hạn ở đây chỉ nằm ở dung lượng của physical memory hoặc kernel cần bộ nhớ cho tác vụ đặc biệt khác.

Đồng bộ kernel buffer của file I/O

Mặc dù các dirty buffer sẽ được kernel ghi vào ổ cứng rất sớm sau khi read() và write() return, đôi khi ứng dụng cần đảm bảo chắc chắn dữ liệu đã được ghi vào ổ cứng từ kernel buffer trước khi tiếp tục. Do vậy, chúng ta cần đồng bộ thủ công (manually) thay vì chờ kernel tự động làm việc đó.

Trước khi tìm hiểu các cách đồng bộ kernel buffer, ta làm quen với hai thuật ngữ (hai loại) về toàn vẹn dữ liệu đồng bộ vào file được định nghĩa bởi tiêu chuẩn SUSv3 là Synchronized I/O

data integrity và Synchronized I/O file integrity. Sự khác nhau giữa hai loại đồng bộ file này nằm ở metadata. Chúng ta hiểu khái quát, metadata là một dữ liệu để mô tả một dữ liệu khác. Metadata của một file là các thông tin của file đó bao gồm owner và group, quyền truy cập file, kích thước file, số hardlink đến file, timestamp chỉ thời gian truy cập file lần cuối, lần thay đổi file cuối cùng, và con trỏ đến dữ liệu file. Chúng ta có thể hiểu là một file bao gồm hai phần: dữ liệu (nội dung của file) và metadata của file đó.

Hiểu một cách cơ bản, Synchronized I/O data integrity chỉ yêu cầu nội dung data của file được đồng bộ giữa tiến trình và file, các metadata không cần thiết có thể không cần ghi vào ổ cứng. Còn Synchronized I/O file integrity có thể nói là tập hợp cha của synchronized I/O data integrity, yêu cầu tất cả nội dung của file bao gồm data và metadata phải được đồng bộ giữa tiến trình gọi và ổ cứng. Ví dụ, nếu bạn gọi write() ghi một buffer vào ổ cứng mà không làm thay đổi kích thước của file thì thuộc tính file size của metadata không thay đổi. Với Synchronized I/O data integrity thì chỉ cần đồng bộ nội dung buffer cần ghi vào file mà không yêu cầu ghi đè file size vào ổ cứng, còn với Synchronized I/O file integrity thì yêu cầu tất cả thông tin của file đó phải được ghi vào ổ cứng.

Các system call điều khiển kernel buffer

Sau khi hiểu được về cơ chế của kernel buffer và đồng bộ kernel buffer xuống file, chúng ta có thể sử dụng các system call để làm các việc này rất dễ dàng. Linux cung cấp các system call đồng bộ thường dùng sau đây:

fsync()

```
#include <unistd.h>

int fsync(int fd);
```

fsync() sẽ đẩy toàn bộ dữ liệu và metadata gắn với mô tả file fd xuống ổ cứng. fsync() return 0 nếu thành công, và -1 nếu xảy ra lỗi. fsync() là system call tương ứng với phương pháp đồng bộ toàn vẹn file I/O.

fdatasync()

```
#include <unistd.h>

int fdatasync(int fd);
```

fdatasync() là system call tương ứng với phương pháp đồng bộ toàn vẹn dữ liệu I/O, nghĩa là chỉ đồng bộ dữ liệu và các thuộc tính metadata cần thiết.

fdatasync() có thể giảm số lần thao tác vào ổ cứng so với fsync(). Ví dụ, nếu ghi vào một file mà dữ liệu thay đổi trong khi kích thước file không đổi, thì fdatasync() chỉ cập nhật dữ liệu vào file, không thao tác với metadata của file đó. Vì data và metadata của file thường nằm ở hai vùng nhớ khác nhau trong ổ cứng, giảm số lần thao tác với metadata của file sẽ làm tăng đáng kể performance của hệ thống.

sync()

```
#include <unistd.h>
```

```
void sync(void);
```

System call sync() sẽ đẩy toàn bộ kernel buffer chứa thông tin file xuống ổ cứng. Lời gọi sync() system call luôn thành công.

Cờ (flag) đồng bộ file khi gọi write()

- O_SYNC

Cờ O_SYNC được dùng ở open() với một file sẽ đồng bộ toàn bộ file đó (data và metadata) xuống ổ cứng sau mỗi lần gọi write():

```
fd = open(pathname, O_WRONLY | O_SYNC);
```

- O_DSYNC và O_RSYNC

Cờ O_DSYNC dùng để xác nhận chỉ đồng bộ dữ liệu (không đồng bộ metadata) sau mỗi lời gọi write(). Cờ O_RSYNC đồng bộ các các yêu cầu cả đọc và ghi file.

Việc sử dụng các cờ đồng bộ file trong open() làm giảm performance của hệ thống rất lớn. Vì vậy trong lập trình Linux system, nếu cần thiết phải gọi đồng bộ nội dung của kernel buffer thì chúng ta nên sử dụng các system call fsync() và fdatasync() thay vì sử dụng các cờ của O_SYNC hay O_DSYNC này.

Kết luận

Cơ chế sử dụng kernel buffer cung cấp giải pháp hiệu quả hơn cho việc thao tác đọc ghi file. Trong một số trường hợp tiến trình cần chắc chắc các thông tin của file đã được cập nhật vào ổ cứng thì có thể sử dụng các system call fsync() hay fdatasync(). Tuy nhiên, bạn không nên lạm dụng các system call này vì có thể làm ảnh hưởng đến performance của hệ thống.