第 7 章

ファイルの読み書き

永続ストレージ上のデータ操作は，メインメモリ上でのデータ操作とは異なります．この章では，典型的な永続データを管理するための方法として，ファイルの読み書きについて説明します．ファイルシステムは，ほとんどの OS が用意している永続データ管理の仕組みです．もちろん，ブロックデバイスを直接扱ったり，他のデバイス専用インターフェースを利用して永続データを操作することも有り得ますが，それはデバイスの性能特性をより生かしたいとか，特殊なインターフェースを使わざるを得ない，など特別な理由があるときに限られます．

ここでは Linux OS のシステムコールを使って説明しますが，各言語で使えるファイル操作ライブラリなどは全部これらのラッパーと思ってもらって良いです．引数をどのように指定するか，返り値をどう解釈するかなど，具体的なシステムコールの使い方については説明していませんので，必要に応じて man を見てください．`man 2 open` と実行すると，システムコール `open()` の manual を読めます．数字の意味は `man man` を参照のこと．

ファイル操作の方法は主に二種類存在します．read/write を使う方法と mmap を使う方法です．

7.1 基本的なシステムコール

共通: `open()`, `close()`

ファイルを open したり close したりします．open してないファイルは読み書きや mmap できません．Open されているファイルはファイルディスクリプタ (通称 fd) で管理します．`open()` に渡す `flags` に指定できる値は一通り眺めておきましょう．`O_APPEND`，`O_TRUNC`，`O_SYNC`，`O_DSYNC`，`O_DIRECT` あたり．

読み書き: `read()`, `write()`, `lseek()`

その名の通り，ファイル上でデータを読み書きします．ブロックデバイスとは異なり，ファイルシステムがよろしくやってくれますので，Byte 単位で任意の size のデータを読み書きできます．ただし，ファイルシステムの下にブロックデバイスがいる場合は，最終的にブロック単位のアクセスになることはお忘れなく．ファイル上の位置を指定するシステムコールが `lseek()` として分離されているので注意が必要です．`lseek()` と `read()`/`write()` が合体したものと見做せる `pread()`/`pwrite()` もあります．その他，関連するシステムコールとして `readv()`, `writev()`,

`preadv()`, `pwritev()` などがあります．

典型的な使い方では read/write はファイルシステムが管理する page cache を通して下位の永続ストレージデバイスにアクセスします．Page cache とは，ページ (4KiB) 単位のメインメモリを用いたブロックデバイスデータのキャッシュ管理機構です．ユーザーランドから page cache の振舞いを細かく制御することは難しいですが，カーネルにヒントを与えることはできます．`posix_fadvise()` を参照ください．DBMS は自分でキャッシュ管理することも多いです．Page cache を使いたくない場合は `O_DIRECT` で open して使います．その場合，後述する liabio を使うことも考えられるでしょう．

永続化: `fsync()`, `fdatasync()`

データの永続化を行うシステムコールです．`write()` 等の書き込み処理を終えた後に，`fsync()` や `fdatasync()` を呼んで，正常終了したことを確認しない限り，書き込んだデータが永続化されている保証はありません．逆に，`fsync()`/`fdatasync()` を呼ばなくても，永続化されている可能性はもちろんありますので，注意してください．`write()` を呼んだ時点でいつ永続化されても文句はいえませんということです．

mmap

`mmap()` システムコールは，open されているファイルの一部 (または全部) の連続領域をプロセスメモリ空間にマップします．`munmap()` はマッピングを開放します．マッピングされたメモリを読み書きすると，ファイルの読み書きができます．内部的に page cache をうまく使ってくれますが，ファイルシステムが用意しているキャッシュ管理アルゴリズムに依存するところが多いので，細かい制御は難しいです．`msync()` システムコールは，変更の永続化を強制します．`msync()` の完了は永続化済みであることを保証しますが，`fsync()` のときと同様にマッピングされたメモリに書き込んだ時点でいつそのデータが永続化されても文句は言えません．

Mmap はファイルのデータ空間をメモリにマップする機能以外に，メモリを確保する用途 (`malloc()` と同様，ただし，内部的な動作は多少違います) でも使えます．特に huge page を確保するために使われます．

> ■コラム: Huge page
>
> Under construction.

Read/write と mmap どちらをどう使うか

単に新しくファイルを open して，先頭から順に書いて，close するのであれば，mmap ではなく `write()` を使うと良いでしょう．Read-only (それ以上変更しないことに決めた) ファイル上で検索などの複雑な読み込み操作をするときは，mmap によるメモリアクセスを使えばコードが単純になると思います．それでも参照データをポインタに変換するなどの手間は必要です．Mmap を使う場

合，変更されたデータの永続化を `msync()` を呼ぶことで保証することができますが，逆に永続化されていないことを保証する仕組みが備わっていないことに注意してください．データがすぐに永続化されては困る場合は，データを別バッファなどに一時的に記録しておいて，マッピングされたメモリに書くのを遅らせるなどの工夫が必要になるでしょう．`write()` を使う場合は，バッファの書き換えと `write()` の呼び出し操作が分離されていますから，ファイルへの反映タイミングをある程度制御できます．

7.2 その他の話題

非同期 IO

Linux では aio という非同期 IO が使えます．同期 IO は対応する read/write システムコールが完了したら IO も完了しています (永続化されているわけではないことに注意) が，非同期 IO では IO の submit と completion 待ちが別の API として分かれています．`open()` した後，aio 専用のシステムコール (そのラッパー) を呼ぶことに変わりはありませんが，`open()` の flags 引数に `O_DIRECT` をつけること，使うバッファが block size に alignment されていることなどが要求されます．興味がある人は libaio というライブラリをインストールして使ってみてください．Linux においては posix aio (`struct aiocb` を使う) も使えますが，libaio (`io_context_t` を使う) の方が良いです．

最近の Linux kernel (5.1) には `io_uring` という新しいインターフェースも加わりました．`io_uring` を使うと IO 毎のシステムコールが不要になり，オーバーヘッドが小さくなる利点があります．DPDK や SPDK (ユーザーランドとデバイス間のやりとりにおいてカーネルを介さない) ほど思い切ったアプローチではありませんが，ユーザーランドとカーネルのやりとりを効率化するという点では近いものを感じます．いずれ非同期 IO のデファクト・スタンダードは `io_uring` になると思われます．

Hole

Linux ファイルシステムの比較的新しい機能として hole があります．文字通り，ファイルに穴 (hole) を空けます．hole は zero データが入っているように見えますが，ファイルのデータブロックとしての実体は確保されていません．hole には使用領域を節約できる，アクセスを高速化できる，という効果があります．`fallocate()` を使って穴を空けます (punch hole)．最小 size や alignment の制約が強いので注意してください．必要な場合，`lseek()` の `whence` 引数に `SEEK_DATA` か `SEEK_HOLE` を指定して hole を検知します．ファイルシステムがサポートしている場合は使えます．

その他 API

`ftruncate()` は，ファイルサイズを変更します．`flock()` はファイルを排他します．同様の用途で `lockf()` や `fcntl()` もありますが，微妙に違います．

メタデータ/ディレクトリの操作

ファイルのメタデータやディレクトリの操作は，別のシステムコール/関数群があります．

- `fstat()`, `stat()`, `lstat()`
- `chmod()`, `chown()`
- `link()`, `symlink()`, `unlink()`, `rename()`
- `mkdir()`, `rmdir()`
- `opendir()`, `closedir()`, `readdir()`, `scandir()` (これらはシステムコールというより glibc 関数)