情報処理システム Ⅱ ・第 10 回

2025 年 6 月 13 日

配布資料の内容

10.1 Unix 系 OS でのファイル入出力

情報実習室の PC や、Windows 環境の WSL[1] にインストールされた Linux は、一般に Unix 系と呼ばれる OS の仲間です。Unix は 1970 年頃、AT&T 社の Bell 研究所で、Ken Thompson と Dennis Ritchie を中心としたグループが開発したオペレーティングシステムです。同じ頃、Dennis Ritchie は、Unix の開発向けに C 言語を設計し、Unix のほとんどの部分が C 言語で記述されるようになりました。その後、Unix は多くの人の手で機能強化されるとともに、たくさんの変種[2]が生まれましたが、1980 年代の中頃から Unix の基本的な仕様を標準化する動きが起こり、POSIX[3] と呼ばれる共通仕様が定められるようになりました。現在の多くの Unix 系 OS は、この POSIX にほぼ準拠したものとなっています。

メモ

ファイル記述子　Unix 系 OS では、ユーザプロセスがファイルや各種のデバイス、ネットワークなどに対して行う入出力処理を、すべて共通の枠組みで行います。この際の手順は以下の通りです。

1. ユーザプロセスは、入出力先となるファイルやデバイスをパス名で指定して、**open** というシステムコールにより入出力の口を**開く**(作成する)[4]。プロセスに作成された入出力の口は**ファイル記述子**と呼ばれる非負の整数で識別します。
2. ユーザプロセスは、ファイル記述子 (非負整数) で入出力先を指定して、**read** や **write** などのシステムコールにより、データの**読み書き**を必要なだけ行う。
3. 入出力処理を終えたユーザプロセスは、不要となったファイル記述子 (非負整数) を **close** というシステムコールで**閉じる**。

[1] Windows Subsystem for Linux

[2] 現在も使用されている Unix 系 OS として、Sun Microsystems 社 (現 Oracle 社) の Solaris、Hewlett Packard 社の HP-UX、IBM 社の AIX、カリフォルニア大学バークレイ校のグループが拡張したバージョンを元に多くの人が機能拡張した FreeBSD や OpenBSD、NetBSD、フィンランドのヘルシンキ大学の学生だった Linus Torvalds が開発を始めて多くの人によって機能強化され現在に至っている Linux、Apple 社の macOS などがあります。

[3] 現在の最新版は IEEE Std 1003.1-2008 (POSIX:2008) となっています。

[4] socket などのシステムコールにより、ネットワークにアクセスする際の通信の口を開く (作成する) こともあります。

ここで、登場した open、read、write、close などのシステムコールは、すべて標準のライブラリ関数として、C 言語から呼び出せるようになっています。

メモ

ファイルへの書き込み　例えば、/home/y240000/data.out という絶対パス名で指定されるファイルに「Hello!」「...」「Bye!」の 3 行からなる文字列を書き込むには

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

int main()
{
    int fd = open("/home/y240000/data.out", O_WRONLY);

    write(fd, "Hello!\n", 7);
    write(fd, "...\n", 4);
    write(fd, "Bye!\n", 5);
    close(fd);
    return 0;
}
```

のように、open、write、close のシステムコールを呼び出します。open の第 2 引数となっている O_WRONLY は、fnctl.h でマクロ定義された定数で、このファイルを書き込み専用で開くことをカーネルに指示するものです。カーネルは、指定されたファイルをファイルシステム内で見つけ出し、そのアクセス保護情報 (保護モード) を調べて、このユーザプロセスが、そのファイルに書き込む権限を持っているかどうかをチェックします。確かにファイルが存在していて、プロセスに書き込み権限がある場合は、カーネルは、このプロセスに新しい入出力の口を作り、そのファイル記述子を関数の戻り値として返します。このファイル記述子が変数 fd に記憶され、その後の write や close の第 1 引数として使用されています。

write は、指定されたファイル記述子の先にあるファイルなどに、第 2 引数で指定されたアドレスから格納されているデータを第 3 引数の長さ (バイト数) だけ書き込みます。最初の

```
write(fd, "Hello!\n", 7);
```

が実行されると、その先頭から「Hello!」とそれに続く改行文字 (合わせて 7 B) が書き込まれます。これにより、このファイルの先頭にあった 7B の古いデータは上書きされます。続く

```
write(fd, "...\n", 4);
```

は、この 7 B に続く 4 B としてファイルに書き込まれます。

```
write(fd, "Bye!\n", 5);
```

も同様です。この例では、1 行ずつ write していますが、3 つをまとめて

```
write(fd, "Hello!\n...\nBye!\n", 16);
```

としてもファイルの内容は同じになります。最後の部分の

```
close(fd);
```

が実行されると、この入出力の口 (ファイル記述子) は閉じられます。結局、ファイルの先頭 16 B のデータが上書きされ、それより後に元からあったータは変更されません。

メモ

open が呼び出された際に、指定したファイルが存在しない場合や、このプロセスがそのファイルに対する書き込みの権限を持っていない場合は、システムコール open が失敗し、この関数の戻り値として -1 が返されます。-1 はファイル記述子としては意味を持ちませんので、続く write や close も失敗することになります。また、write は、書き込みに成功したデータの大きさ (バイト数) を戻り値として返します。正常に書き込みができない場合は -1 を戻します。補助記憶装置に十分空き容量がない場合などは、write が失敗することがありますので、注意が必要です。

このように、システムコールは常に成功するとは限りませんから、先のプログラムは次のようにするのが一般的です[5]。

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

int main()
{
    int fd = open("/home/y240000/data.out", O_WRONLY);

    if (fd < 0) {
        printf("ファイルが開けません\n");
        return 1;
    }
    if (write(fd, "Hello!\n", 7) != 7) {
        printf("書き込みに失敗しました\n");
        return 1;
    }
    if (write(fd, "...\n", 4) != 4) {
        printf("書き込みに失敗しました\n");
        return 1;
    }
    if (write(fd, "Bye!\n", 5) != 5) {
        printf("書き込みに失敗しました\n");
        return 1;
    }
```

[5] このプログラムでは、printf を使ってエラーメッセージを出力していますが、通常は、次回に解説するように、fprintf を使って「標準エラー出力」と呼ばれるファイル記述子へ出力します。

```
    close(fd);
    return 0;
}
```

このプログラムでは、指定したファイルが存在していないと open が失敗してしまいますが、open の引数を次のようにして、そのような場合に自動的に新しくファイルを作成することもできます。

```
int fd = open("/home/y240000/data.out", O_WRONLY | O_CREAT, 0644);
```

O_CREAT が、ファイルの自動作成の指示で、WR_ONLY とのビット毎の論理和をとって、open の第 2 引数としてカーネルに伝えています[6]。

また、open の第 2 引数を

```
int fd = open("/home/y240000/data.out", O_WRONLY | O_TRUNC);
```

のようにすると、ファイルがすでに存在している場合に、古い内容をすべて削除し、ファイルの大きさを 0 にしてから新たに書き込みを始めることもできます[7]。

メモ

ファイルの読み出し　ユーザプロセスがファイルの内容を読みたい場合は、そのファイルを読み込み用に open しておき、read というシステムコールを使用して内容を読み込みます。次は、絶対パス名 /home/y240000/data.in で指定されるファイルを /home/y240000/data.out というファイルへコピーするプログラムです。コピー先の /home/y240000/data.out がまだ存在していなければ自動的に作成されます。すでに存在していれば、その古い内容は消去されます。

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

int main()
{
    char buf[1000];
    int in, out, size;

    in = open("/home/y240000/data.in", O_RDONLY);
    out = open("/home/y240000/data.out", O_WRONLY|O_CREAT|O_TRUNC, 0644);

    while (1) {
        size = read(in, buf, 1000);
        if (size < 0) {
            printf("読み込みに失敗しました\n");
            return 1;
```

[6]「0644」は、新たに作成されるファイルの保護モードを 8 進表記したものです。

[7]O_TRUNC は O_CREAT と組み合わせることもできます。標準入出力ライブラリ関数の fopen を使った fopen("...", "w") という呼び出しは、open("...", O_WRONLY|O_CREAT|O_TRUNC, 0666) というシステムコールを行います。

```
            }
            if (size == 0)
                break;
            if (write(out, buf, size) != size) {
                printf("書き込みに失敗しました\n");
                return 1;
            }
        }
        close(in);
        close(out);
        return 0;
    }
```

コピー元のファイルを開いている open の第 2 引数 O_RDONLY が、このファイルを読み込み専用で開きなさいというカーネルへの指示です。その内容の読み込みは read というシステムコールで行っています。read の第 1 引数で指定された読み込み口 (ファイル記述子) から、第 2 引数で指定されたアドレスに、最大、第 3 引数で指定された大きさ (バイト数) だけ、データが読み込まれます。read は実際に読み込むことのできた大きさ (バイト数) を戻り値として返します。read を何度も呼び出すと、まだ読み込んでいない部分が次々に読み込まれます。読み込んでいるファイルの終端まで来ると、read は 0 を戻り値として返します。何らかの問題が発生して読み込みに失敗した場合は、−1 が返されます。

メモ

10.2 Unix 系 OS における入出力関係の代表的なシステムコール

ユーザプログラムはシステムコールによりカーネルの提供する機能を使用します。その際、ソフトウェア割り込みを発生させる機械語命令を実行する必要がありますが、第 8 回に解説したように、多くの OS では、C 言語などの高級言語で書かれたプログラムから容易にシステムコールを行うことができるように、システムコールを行うサブルーチン群 (C 言語の場合は関数群) を納めたライブラリを提供しています。このようなライブラリをアプリケーションプログラムと結合することで、関数呼び出しの形でカーネルの提供する機能を使用することができます。このとき、一つのシステムコールが一つの関数に対応することもあれば、いくつかのシステムコールを組み合わせて、一つの関数が実現される場合もあります。

以下は、Linux などの Unix 系 OS がライブラリ関数として提供している入出力関係の代表的なシステムコール群です[8]。これらは、(現行の) どの Unix 系 OS でも使用することができます。また、OS によっては、システムコールに直接対応しておらず、他のシステムコールを使って実現され

[8] この表のようにソフトウェア的な部品の間 (ここではカーネルとアプリケーションプログラムの間) をソースプログラムレベルでつなぐための約束ごと定めたものを Application Programming Interfaces (API) と呼びます。この科目で紹介するのは、Unix 系 の各 OS カーネルの API のほんの一部となります。

ている関数となっていることもあります。

表中の関数プロトタイプに現れている uid_t、gid_t などの型は OS によって異なります。たとえば、情報実習室の Linux 環境 (64bit) では、これらは次のように定義されています。

```
typedef unsigned int uid_t;
typedef unsigned int gid_t;
typedef unsigned int mode_t;
typedef unsigned long size_t;
typedef unsigned int socklen_t;
```

ファイル管理関連

`int open(char *path, int flags, mode_t perm)`	path で指定したファイルを flags で指定したモードで開き (オープンし)、そのファイル記述子を戻す。flags では、読み込み/書き出し/読み書きのいずれであるか、ファイルが存在しない場合は作成するかどうか、書き込みの際に常にファイルの末尾に書き込むかどうか、などを指定する。ファイルが新たに作成される場合、その保護モードは perm となる。
`int read(int fd, void *buf, size_t size)`	ファイル記述子 fd から、最大 size バイトを、アドレス buf から始まる領域に読み込み、実際に読み込むことのできたバイト数を戻す。
`int write(int fd, void *buf, size_t size)`	アドレス buf から始まる領域のデータを、ファイル記述子 fd へ最大 size バイトを書き出し、実際に書き出すことのできたバイト数を戻す。
`int close(int fd)`	ファイル記述子 fd を閉じる。
`int link(char *path, char *new)`	path で指定したファイルを、新たなパス名 new としてリンクする。
`int unlink(char *path)`	path で指定したファイルを削除する。
`int symlink(char *path, char *new)`	path というパス名へのシンボリックリンクを、new というパス名で作成する。
`int chmod(char *path, mode_t perm)`	path で指定したファイルの保護モードを perm に変更する。
`int chown(char *path, uid_t uid, gid_t gid)`	path で指定したファイルの所有者 (ユーザ ID) を uid に、グループ ID を gid に変更する。
`int mkdir(char *path)`	path で指定したディレクトリを作成する。
`int rmdir(char *path)`	path で指定したディレクトリを削除する。

メモ

デバイス管理関連

`int ioctl(int fd,` `  int req, ...)`	ファイル記述子 fd で指定したデバイスを、req, ... で指定したように制御する。引数 req, ... の意味はデバイスによって異なる。

ネットワーク通信関連

`int socket(int domain,`
`  int type,`
`  int protocol)`
	domain、type、protocol で指定したソケット (通信の端点) を作成し、そのファイル記述子を戻す。

`int bind(int fd,`
`  struct sockaddr *addr,`
`  socklen_t addrlen)`
	ファイル記述子 fd で指定したソケットに、addr と addrlen で指定したアドレス (IP アドレスなど) を割り当てる。

`int connect(int fd,`
`  struct sockaddr *addr,`
`  socklen_t addrlen)`
	ファイル記述子 fd で指定したソケットを、addr と addrlen で指定したアドレス (IP アドレスなど) を持つ通信の端点へ接続する。

`int listen(int fd,` `  int backlog)`	ファイル記述子 fd で指定したソケットを接続待ち状態にする。接続要求の待ち行列の最大長を backlog で指定する。
`int accept(int fd)`	ファイル記述子 fd で指定した接続待ち状態のソケットへの次の接続要求を受理し、その接続の通信のための新しいソケットを作成し、そのファイル記述子を戻す。

一般的な入出力関連

`int dup(int fd)`	ファイル記述子 fd の複製を作成して戻す。
`int dup2(int fd, int new)`	ファイル記述子 fd を複製したものをファイル記述子 new とする。
`int fcntl(int fd,` `  int cmd, ...)`	ファイル記述子 fd の取り扱いに関する各種の設定を、cmd, ... で指定する。

`int select(int nfds,`
`  fd_set *rfds,`
`  fd_set *wfds,`
`  fd_set *efds,`
`  struct timeval`
`  *timeout)`
	rfds、wfds、efds で指定したファイル記述子の 3 つの集合に含まれる (最大 nfds 個までの) ファイル記述子のいずれかが、それぞれ、読み込み可能、書き出し可能、例外 (エラー) 発生状態になるのを最大 timeout で指定した時間だけ待ち、そのような状態になったファイル記述子の集合を *rfds、*wfds、*efds に書き込み、その数を戻り値として戻す。

情報処理システム Ⅱ ・第 10 回・終わり