TK-850/JH3U-SP
LCD 表示 サンプルプログラム
操作説明書

（LCD 有り版向け）

第 1.0 版

テセラ・テクノロジー（株）

目次

1. はじめに

1.1. 目的

本書は、TK-850/JH3U-SP(以降ボード)に付属される、「LCD 表示 サンプルプログラム」の操作説明等を記載したものです。

1.2. 動作環境

TK-850/JH3U-SP ボード （LCD 有り版）
付属 USB メモリ
付属 AC アダプタ電源供給

1.3. 開発環境

統合環境: PM+ V6.31
C コンパイラ: CA850 V3.20
デバッガ: ID850QB V3.50
デバイスファイル:uPD70F3769 (V1.00)

1.4. 制限事項

USB ホストドライバが含まれている「**libJx3Usp.a**」は本ボード専用です。お客様が開発されたボードで使用することは出来ません。

2. 動作概要

ボードに接続された USB メモリから、複数のビットマップファイル(BMP ファイル)を読み込んで、LCD に順次表示します。

2.1. 動作仕様

・BMP ファイル名は 8.3 形式
・ファイルの保存場所は、ルートディレクトリとする（サブフォルダには入れない）
・BMP ファイルのフォーマットは 1bit、4bit、8bit、24bit、32bit に対応
・表示画像は左上から、最大で横 320 ピクセル、縦 240 ピクセルを表示。それ以上(画面外)は無視されます。
・表示されるビットマップファイルは、最大 20 枚

2.2. 操作手順

(1) 付属 USB メモリへ添付の BMP ファイルを格納します。(フォルダに格納しないでください。)

(2) USB メモリをボードに接続し、AC アダプタから電源を供給します。
(3) PM+で「Jx3Usp_LCDsample¥NEC_project¥Jx3Usp_LCDsamplle.prw」ワークスペースを開き、デバッガを起動して、プログラムを RUN させます。
10 秒おきに BMP ファイルが読み込まれ、LCD に表示されます。

注意
USB2 を使用した USB ホスト接続を行う場合、および LCD 表示を行う場合には必ず AC アダプタからの電源供給を使用してください。

3. サンプルプログラム概要

3.1. ファイル構成

以下に、サンプルプログラムのファイル構成を示します。

```
C:¥TK850
  |
  |---Jx3Usp_LSDsample
        |---include    : ヘッダファイル
        |    |---main.h
        |    |---apl.h
        |    |---vramop.h
        |    |---bmp2lcd.h
        |
        |---lib
        |    |---inc850
        |    |    |---cfs  : ファイル・システム用ヘッダファイル群
        |    |    |---msc  : USB ホストドライバ用ヘッダファイル群
        |    |
        |    |---lib850e
        |         |---libJx3Usp.a     : ファイル・システム、USB ライブラリ
        |
        |---NEC_project  : プロジェクトファイル関連
        |    |---Jx3Usp_LCDsamplle.prw   : プロジェクトワークスペース
        |
        |---src   : ソースファイル
             |---main.c
             |---apl.c
             |---initialize.c
             |---bmp2lcd.c
             |---vramop.c
             |---fstime.c
```

(1) main.c/h

メイン関数。および ROM 化処理。

(2) initialize.c

ボード初期化処理。

(3) apl.c/h

サンプルプログラムのメイン処理。USB から BMP ファイル名を取り出し、10 秒カウントごとに順次表示処理を実行する。

ヘッダ内の FILE_MAX 値を変更することで、表示する BMP ファイルの数を変更できます。また、WAIT_100MS 値を変更することで、表示後のウェイト時間を変更できます。

(4) bmp2lcd.c/h

指定された BMP ファイル名から、BMP ファイルを解析し LCD への表示処理を実行する。

(5) vramop.c/h

VRAM への操作処理群。

ヘッダ内の DISPLAY_DIRECTION 値を変更することで、画面表示の上下左右を反転します。

(6) fstime.c/h

ファイル作成、更新時のタイムスタンプ生成処理(本サンプルプログラムでは未使用)。

3.2. ソフトウェア構成

以下にサンプルプログラムのソフトウェア構成を示します。

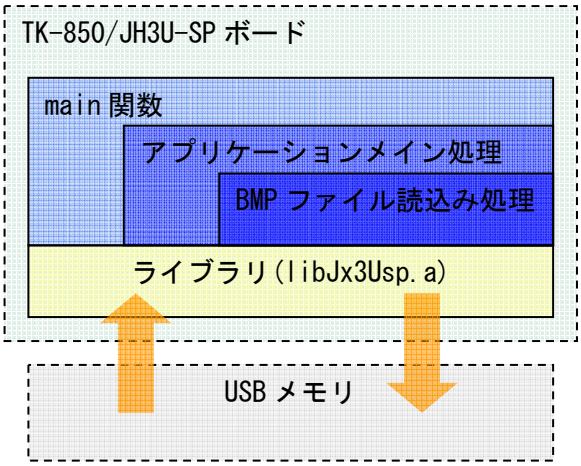

図 3.1 ソフトウェア構成

3.3. 処理フロー

以下にサンプルプログラムの大まかな処理の流れを示します。

3.3.1. main 関数

USB メモリの接続をチェックし、接続されて、且つ Enumeration が終了していたら、アプリケーションメイン処理へ移行します。接続が正常にいかないときは、正常になるまでループします。

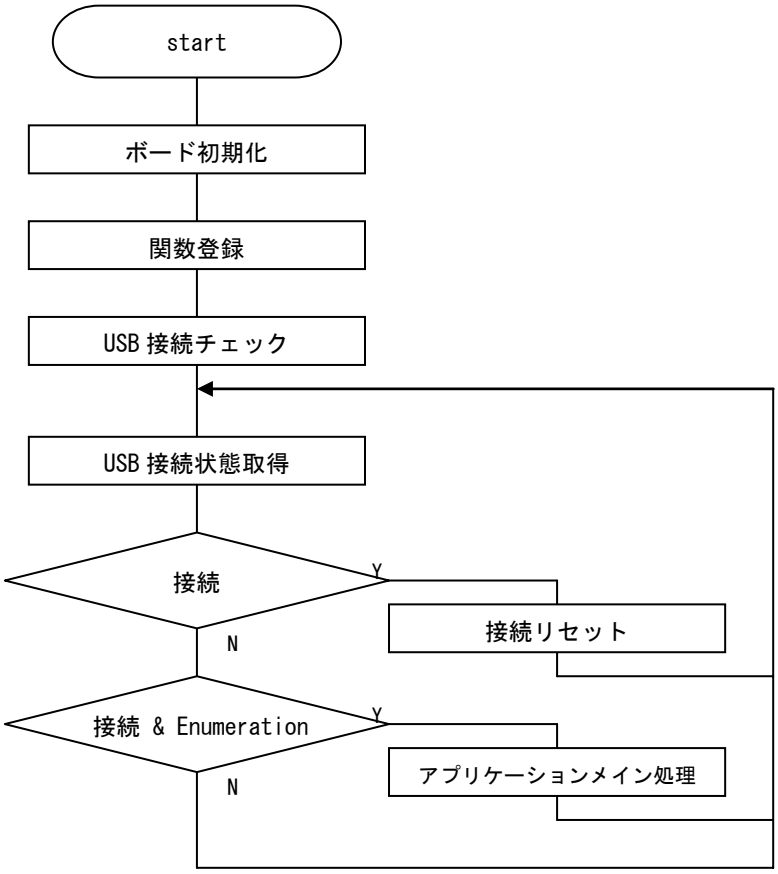

図 3.2 main 関数処理フロー

3.3.2. apltask 関数(アプリケーションメイン処理)

USB メモリから、複数の BMP ファイル名を取得します。USB メモリをマウント後、ディレクトリオープン(opendir)でデバイス直下(root)のディレクトリ内を取得します。リードディレクトリ(readdir)を繰り返し、ファイル名を取得し、BMP ファイルの場合(拡張子が”BMP”の場合)、このファイル名を退避します。この操作をディレクトリ内のファイルがなくなるまで行うか、最大カウント数(デフォルトで 20 枚)まで繰り返します。

次に、ファイル名を順次 BMP 読込み表示処理へ渡します。また、処理がうまくいった場合、10 秒間の wait 処理を行い、その後、次の BMP ファイル名を、BMP 読込み表示処理へ渡します。サンプルプログラムはこの処理をループします。

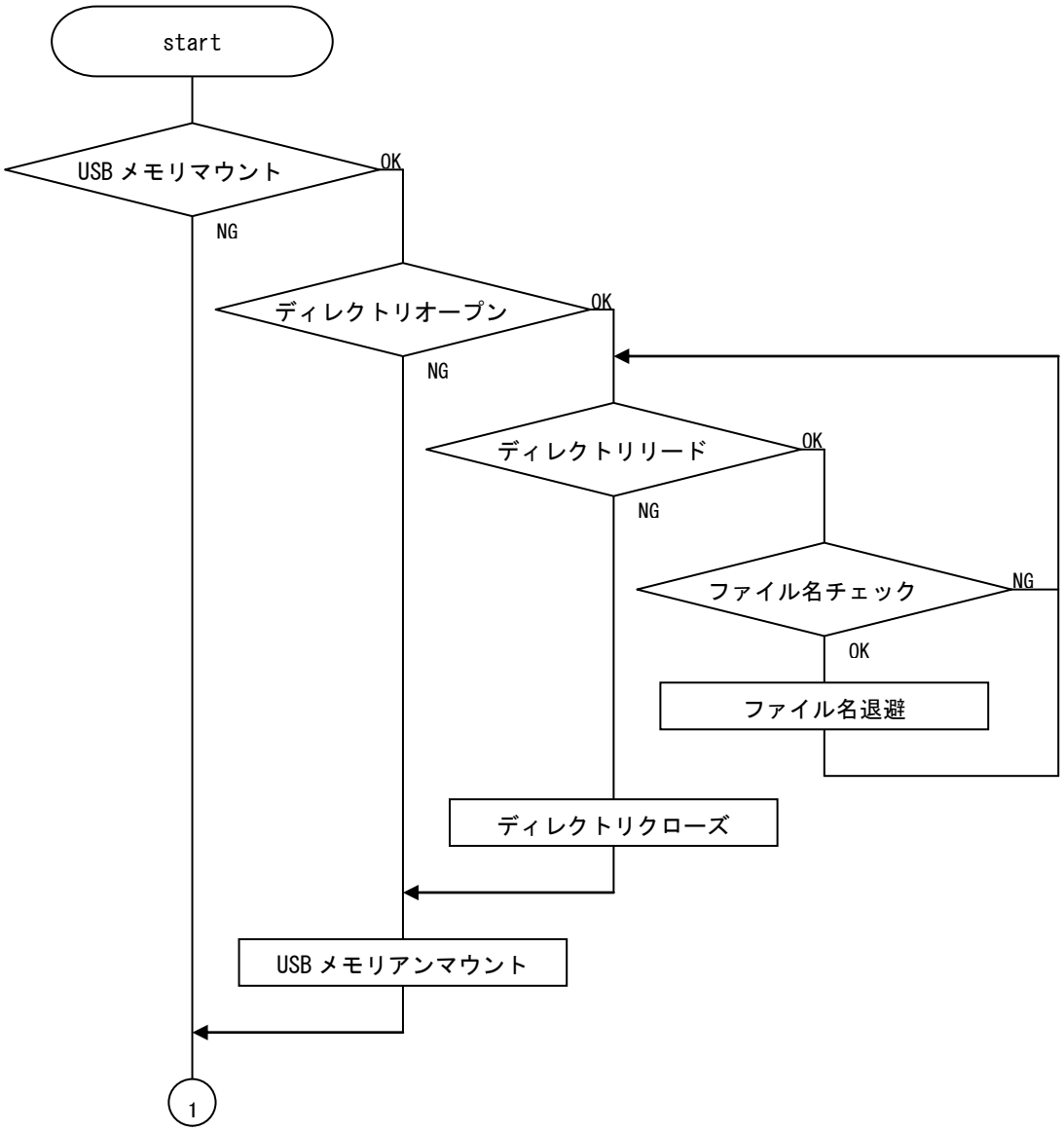

図 3.3 アプリケーションメイン処理フロー①

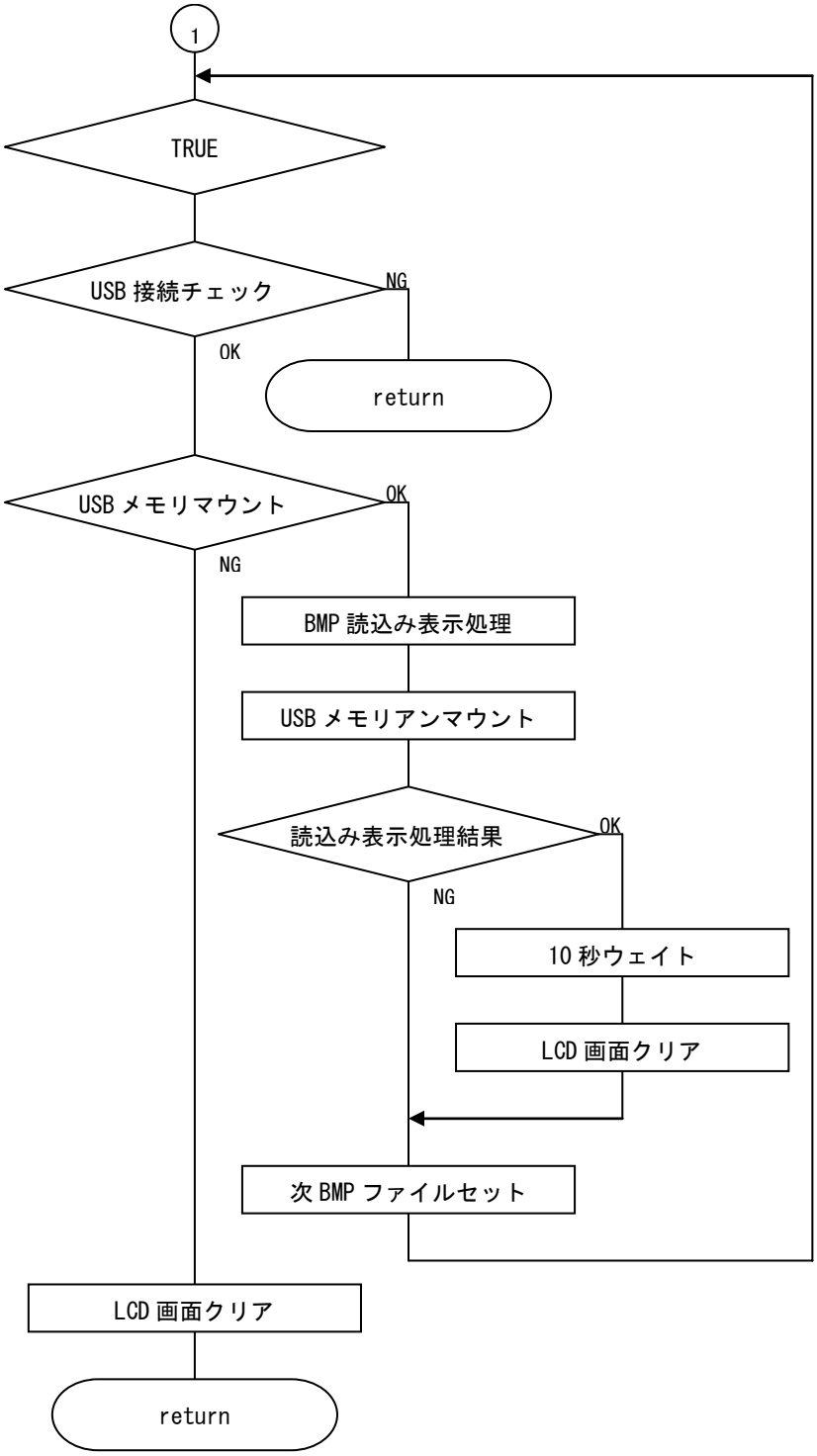

図 3.4 アプリケーションメイン処理フロー②

3.3.3. bmp2lcd 関数(BMP ファイル読込み表示処理)

アプリケーションメイン処理から、表示するファイル名を受け取り、ファイルオープン処理(open)を実行します。その後、ファイルのヘッダ情報を読み込み BMP ファイルかどうかのチェック、およびファイルのサイズに対するデータのオフセットを取得します。次に BMP 情報を取得、このとき、画像のサイズ、色の深度を確認し、必要であれば、カラーパレットを取得します。最後に、取得した情報を元に、BMP データを読込み、1 ピクセルずつ LCD へ表示していきます。

本関数で、パラメータに表示場所(x、y)の指定と、表示方向(上下、左右反転)の指定が可能です。

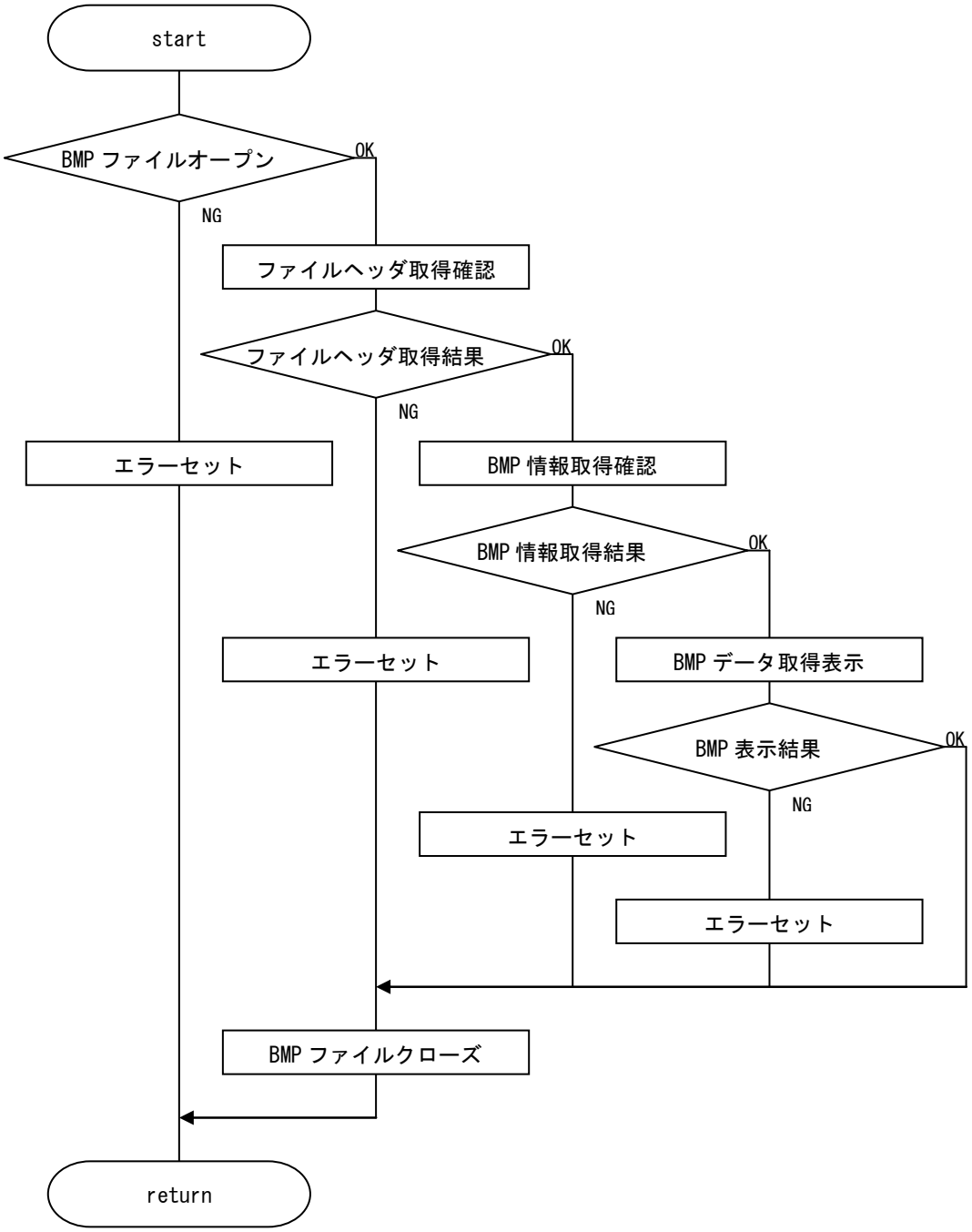

図 3.5 BMP 読込み表示処理

3.3.4. LCD 画面表示

LCD 画面は、1 ピクセルに対して 2 バイトの色データを持ち、このデータを VRAM へ書き込むことで描画します。

(1) BMP 解析により、取得した色のパターンを以下のように処理します。
色データは red、green、blue のそれぞれ 1 バイトになります。
使用できる色データは 5 ビットなので、下位 3 ビットを削ります。

```
blue   >>= 3;
green  >>= 3;
red    >>= 3;

// 'RRRRRGGGGGOBBBBB'(16bit) の並びになる
pattern = (red << 11) | (green << 6) | (blue);

// pattern を VRAM へ書き込む(vp は VRAM へのポインタ)
*vp = pattern;
```

3.4. 実行モジュール作成

サンプルプログラム・ソース内の NEC_Project 内のプロジェクトファイル Jx3Usp_LCDsamplle.prw を PM+で開いて、ビルドを実行してください。

4. USB メモリアクセス

USB メモリへのアクセスには、V850 用のファイル・システム(CompactFS)を使用しています。

4.1. ファイル・システム概要

4.1.1. 機能制限

ファイル・システムには以下の機能制限があります。

- FAT12、FAT16、FAT32 に対応します。
- 2GB を越えるファイルにはアクセスできません。
- ファイル名は、8.3 形式ファイル名での対応になります。また、ファイル名は大文字、小文字を区別しません。本ファイル・システムで作成したファイル名は、全て大文字になります。
- パーティション分割に対応しません。
- マルチタスクに対応しません。

4.1.2. パス名

ファイル・システムで使用するパス名は、以下の形式(フルパス)で指定してください。

/(ドライブ名)/(ファイル名)
/(ドライブ名)/(ディレクトリ名)/(ファイル名)
/(ドライブ名)/(ディレクトリ名)/・・・/(ディレクトリ名) /(ファイル名)

- ドライブ名には mountfs()で指定した名前を記述します。
- デイレクトリセパレータには、'/' 以外を指定できません。Windows で使用されている、'¥' やバックスラッシュは指定できません。
- パス名には、以下の文字は使用できません。
  " * + , . / : ; < = > ? [¥] |
- 相対パスは指定できません。

4.2. アクセス方法

USB メモリをマウントして、ファイルをオープンする例を示します。

(1) デバイスドライバで用意されている FS_DRIVE をマウントします。
以下は、マウントしたデバイスに"USB"というドライブ名をつけています。

```
extern FS_DRIVE  FileDriverDesc;  // デバイスドライバで定義されている
```

```
return_value = mountfs( &FileDriverDesc, "USB" );
```

以降は、デバイス内のファイルを指定する際は、先頭に"USB"をつけてアクセスします。

(2) ファイルをオープンします。

FS_FILE でファイルポインタを作成し、デバイス内の直下にある"test.bmp"ファイルを読込み専用でオープンします。

```
FS_FILE  fp;  // ファイルポインタ

return_value = open( &fp,  "/USB/test.bmp", O_RDONLY );
```

以降のファイルアクセスは、ファイルポインタ fp を使用して行います。

4.3. ファイル・システム API

以下に、ファイル・システムの API 一覧を示します。

API	説明
open	ファイルを開きます。
close	ファイルを閉じます。
read	ファイルの内容を読み込みます。
write	ファイルにデータを書き込みます。
lseek	ファイルの読み書き位置を移動します。
stat	ファイルの情報を取り出します。
chmod	ファイルの属性を変更します。
unlink	ファイルを削除します。
rename	ファイル名を変更します。
opendir	ディレクトリを開きます。
readdir	ディレクトリ内のファイル名を得ます
closedir	ディレクトリを閉じます
mkdir	ディレクトリを作成します。
rmdir	ディレクトリを削除します。
mountfs	ファイル・システムをマウントして、アプリケーションが使用できるようにします。
umountfs	ファイル・システムをマウント状態から開放します。

4.3.1. open

【概要】

指定されたファイルを開きます。

【C 言語形式】

long open(FS_FILE* *file*, const char* *pathname*, long *flags*);

【パラメータ】

I/O	パラメータ	説明
O	FS_FILE* *file*	FS_FILE 構造体を指定します。この構造体の内容は、close()するまで使用されます。
I	const char* *pathname*	開くファイルのパス名を指定します。
I	long *flags*	オープンモードを指定します。

【機能】

pathname に指定されたファイルを開きます。
pathname にはフルパスを指定します。

flags には、O_RDWR、O_RDONLY、O_WRONLY のいずれか1つを指定する必要があります。

O_RDWR: ファイルを読み書きモードでオープンします。
O_RDONLY: ファイルを読込み専用でオープンします。
O_WRONLY: ファイルを書き込み専用でオープンします。

さらに *flags* には以下も指定可能です。

O_APPEND: ファイルを追加モードでオープンします。このモードでは、ファイルの write() のたびに、ファイルポインタをファイルの最後に移動します。O_APPEND を指定しない場合、ファイルポインタはファイルの先頭を示します。
O_CREAT: ファイルを新規作成します。ファイルが既に存在した場合は、その内容が失われます。
O_EXCL: これを単独では使用できません。O_CREAT と一緒に使用された場合、ファイルが既に存在したときにはエラーを返します。

【戻り値】

FS_OK: 正常終了。
FS_ENODEV: *pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。

FS_EACCESS: *pathname* で指定したファイルが書き込み禁止であるのに、O_RDWR、O_WRONLY のいずれかが指定されています。あるいは、ドライブが書き込み禁止のときに、O_RDWR、O_WRONLY のいずれかが指定されています。

FS_EEXIST: *pathname* で指定したファイルが既に存在しているのに、O_CREAT と O_EXCL が使用されました。

FS_ENOENT: *file* あるいは *pathname* に指定されたディレクトリが存在しません。

FS_ENAMETOOLONG: *pathname* が長すぎます。

FS_EPARAM: *file* あるいは *pathname* が NULL です。

FS_EINVAL: *pathname* に使用できない文字が含まれています。

FS_EIO: I/O エラーが発生しました。

FS_ ENOSPC: ドライブに十分な空きがありません（ディレクトリエントリを作成するスペースがありません）。

FS_ENOTREADY: ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.2. close

【概要】

ファイルを閉じます。

【C 言語形式】

long close(FS_FILE* *file*);

【パラメータ】

I/O	パラメータ	説明
I	FS_FILE* *file*	open()で指定した FS_FILE 構造体を指定します。

【機能】

・ *file* で指定されたファイルをクローズします。

【戻り値】

FS_OK: 正常終了。

FS_EPARAM: *file* が NULL です。

4.3.3. read

【概要】

ファイルの内容を読み込みます。

【C 言語形式】

long　read(FS_FILE* *file*, void* *buf*, long *count*);

【パラメータ】

I/O	パラメータ	説明
I	FS_FILE* *file*	open()で指定した FS_FILE 構造体を指定します。
O	void* *buf*	読み込んだデータを保持する領域のポインタを指定します。
I	long *count*	読み込むサイズを指定します。

【機能】

- ファイル *file* の現在のファイルポインタの位置から、*count* バイトを、*buf* で指定されるアドレスに読み込みます。
- 読み込みに成功した場合、ファイルポインタを読み込んだバイト数だけ進めます。
- 読み込みに成功した場合、読み込んだサイズを返します。
- *count* で指定されたサイズを読み込む前にファイルの終端に達した場合は、終端までを *buf* で指定されるアドレスに読み込み、そのバイト数を返します。
- 読み込みに失敗した場合には、戻り値に負の整数を返します。

【戻り値】

正常終了の場合には、読み込んだデータサイズ（正の整数）を返します。読み込みに失敗した場合には、負の整数（下記、FS_で始まるエラーコード）を返します。

FS_ENOTOPEN:	**オープンされていません。**
FS_EACCESS:	**書き込み専用モードでオープンされています。**
FS_EPARAM:	***file* あるいは *buf* が NULL です。または、*count* が0以下です。**
FS_EIO:	**I/O エラーが発生しました。**
FS_ENOTREADY:	**ドライブが準備されていません（媒体が装てんされていないなど）。**

4.3.4. write

【概要】

ファイルにデータを書き込みます。

【C 言語形式】

long write(FS_FILE* *file*, const void* *buf*, long *count*);

【パラメータ】

I/O	パラメータ	説明
I	FS_FILE* *file*	open()で指定した FS_FILE 構造体を指定します。
I	const void* *buf*	書き出すデータを保持する領域のポインタを指定します。
I	long *count*	書き出すサイズを指定します。

【機能】

- write() は *buf* で指定されるアドレスから *count* バイトを、ファイル *file* のファイルポインタの位置から書き込みます。
- 書き込みに成功した場合、書き込んだサイズを返します。
- 書き込んだデータは、セクタ単位の読み書きの切り替えが生じて初めて装置に書き込まれます。このため、書き込みの途中でプログラムを停止した場合には、データが失われる場合があります（他のファイルへの影響はありません）。

【戻り値】

書き込んだサイズを返します。write() に失敗した場合には、負の整数（下記、FS_で始まるエラーコード）を返します。

FS_OK:	正常終了。
FS_ENOTOPEN:	オープンされていません。
FS_EACCESS:	読み込み専用モードでオープンされています。あるいはドライブが書き込み禁止のときに、O_RDWR、O_WRONLY のいずれかが指定されました。
FS_EFBIG:	ファイルサイズの制限を越えて書き込もうとしました。
FS_ENOSPC:	ドライブに十分な空きがありません。
FS_EPARAM:	*file* あるいは *buf* が NULL です。または、*count* が0以下です。
FS_EIO:	I/O エラーが発生しました。
FS_ENOTREADY:	ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.5. lseek

【概要】

ファイルポインタを移動します。

【C 言語形式】

long lseek(FS_FILE* *file*, long *offset*, int *whence*);

【パラメータ】

I/O	パラメータ	説明
I	FS_FILE* *file*	open()で指定した FS_FILE 構造体を指定します。
I	long *offset*	移動するオフセット値を指定します。負の値を指定することもできます。
I	int *whence*	オフセットの原点を指定します。

【機能】

- lseek() は、ファイル *file* のファイルポインタを、 以下に示す *whence* の値に基づき *offset* の位置へ変更します。
- *whence* には以下が指定できます。

FS_SEEK_SET: ファイルポインタはファイルの先頭から *offset* バイトの位置に設定されます。

FS_SEEK_CUR: ファイルポインタは現在位置に *offset* バイトを足した位置になります。

FS_SEEK_END: ファイルポインタはファイルのサイズに *offset* バイトを足した位置になります。

【戻り値】

新しいファイルポインタの位置を、ファイルの先頭からのバイト数で返します。lseek() に失敗した場合には、負の整数（下記、FS_で始まるエラーコード）を返します。

FS_ENOTOPEN: オープンされていません。

FS_EINVAL: 新たなファイルポインタが、ファイル・オフセットが負になってしまうか、ドライブの終端を越えています。

FS_EPARAM: *file* あるいは *buf* が NULL です。または *whence* が SEEK_SET, SEEK_CUR, SEEK_END のどれでもありません。

FS_EACCESS: ファイルが書き込み禁止のとき、かつ、ファイルの終端以降が指定された場合か、ドライブが書き込み禁止です。

FS_EIO: I/O エラーが発生しました。

FS_ENOTREADY: ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.6. stat

【概要】

ファイルの情報を取得します。

【C 言語形式】

long stat(const char* *pathname*, FS_STAT* *stat*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *pathname*	取得したいファイルのパス名を指定します。
O	FS_STAT* *stat*	ファイルの情報を格納する FS_STAT 構造体を指定します。

【説明】

- stat() は *pathname* で指定されたファイルの状態を取得して *stat* へ格納します。
- *pathname* にはフルパスを指定します。

【戻り値】

FS_OK:	正常終了。
FS_ENODEV:	*pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。
FS_ENAMETOLONG:	*pathname* が長すぎます。
FS_EPARAM:	*pathname* あるいは *stat* が NULL です。あるいは、パス名に使用できない文字が含まれています。
FS_EINVAL:	*pathname* に使用できない文字が含まれています。
FS_ENOENT:	ファイルが存在しません。
FS_ENOTREADY:	ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.7. chmod

【概要】

ファイルの属性を変更します。

【C 言語形式】

long chmod(const char* *pathname*, mode_t *mode*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *pathname*	属性を変更したいファイルのパス名を指定します。
I	mode_t *mode*	変更する属性を指定します。

【機能】

- chmod() は *pathname* で指定されたファイルの属性を変更します。
- *pathname* にはフルパスを指定します。
- 属性(*mode*)は、次の値の *or* をとったもので指定します。指定しなかった属性はクリアされます。

  ATTR_READ_ONLY: 書き込み禁止
  ATTR_HIDDEN: 隠しファイル
  ATTR_SYSTEM: システムファイル
  ATTR_ARCHIVE: アーカイブ

【戻り値】

FS_OK:	正常終了。
FS_ENODEV:	*pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。
FS_ENAMETOOLONG:	*pathname* が長すぎます。
FS_EPARAM:	*pathname* が NULL です。あるいは、*pathname* に使用できない文字が含まれています。
FS_EINVAL:	*pathname* に使用できない文字が含まれています。
FS_ENOENT:	ファイルが存在しません。
FS_EACCESS:	ファイルが書き込み禁止です。または、ドライブが書き込み禁止です。
FS_EIO:	I/O エラーが発生しました
FS_ENOTREADY:	ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.8. unlink

【概要】

ファイルを削除します。

【C 言語形式】

long unlink(const char* *pathname*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *pathname*	取得したいファイルのパス名を指定します。

【説明】

- unlink() はファイル・システム上の *pathname* で指定されたファイルを削除します。
- *pathname* にはフルパスを指定します。

【戻り値】

FS_OK:	正常終了。
FS_ENODEV:	*pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。
FS_ENAMETOOLONG:	*pathname* が長すぎます。
FS_EPARAM:	*pathname* が NULL です。あるいは、使用できない文字が含まれています。
FS_EINVAL:	*pathname* に使用できない文字が含まれています。
FS_ENOENT:	ファイルが存在しません。
FS_EACCESS:	ファイルが書き込み禁止です。または、ドライブが書き込み禁止です。
FS_EDIR:	*pathname* に指定されたものが、ディレクトリ名です。
FS_EIO:	I/O エラーが発生しました
FS_ENOTREADY:	ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.9. rename

【概要】

ファイル名を変更します。

【C 言語形式】

long rename(const char* *oldpath*, const char* *newpath*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *oldpath*	変更前のファイルのパス名を指定します。
I	const char* *newpath*	変更後のファイルのパス名を指定します。

【機能】

- rename() はファイルの名前を変更します。
- *oldpath*、*newpath* にはフルパスを指定ます。
- ディレクトリ間の移行は行わないため、*oldpath* と *newpath* のディレクトリ部分を同じにしなければエラーになります。
- *newpath* のファイルが既に存在する場合は、エラーになります。

【戻り値】

FS_OK:	正常終了。
FS_ENODEV:	*oldpath* または、*newpath* に指定されたドライブが見つかりません。あるいはドライブを指定していません。
FS_ENAMETOLONG:	*oldpath* または、*newpath* が長すぎます。
FS_EPARAM:	*oldpath* または *newpath* が NULL です。
FS_EINVAL:	*oldpath* または *newpath* に使用できない文字が含まれています。
FS_EXDIR:	*oldpath* と *newpath* が同じディレクトリではありません。
FS_ENOENT:	*oldpath* が存在しません。
FS_EACCESS:	*newpath* が書き込み禁止です。またはドライブが書き込み禁止です。
FS_EIO:	I/O エラーが発生しました。
FS_ENOTREADY:	ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.10. opendir

【概要】

ディレクトリを開きます。

【C 言語形式】

long opendir(FS_DIR* *dir*, const char* *name*);

【パラメータ】

I/O	パラメータ	説明
I	FS_DIR* *dir*	オープン状態を保持する FS_DIR 構造体を指定します。

I	const char* *pathname*	開くディレクトリ名を指定します。

【機能】

- opendir() はディレクトリをオープンし、その状態を *dir* に保持します。
- *pathname* にはフルパスを指定します。

【戻り値】

FS_OK:	**正常終了。**
FS_ENODEV:	***pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。**
FS_ENAMETOLONG:	***pathname* が長すぎます。**
FS_ENOENT:	**ディレクトリが存在しません。または *pathname* が空文字列です。**
FS_EPARAM:	***dir* あるいは *pathname* が NULL です。**
FS_EINVAL:	***pathname* に使用できない文字が含まれています。**
FS_ENOTREADY:	**ドライブが準備されていません（媒体が装てんされていないなど）。**

4.3.11. readdir

【概要】

ディレクトリ情報を取得します。

【C 言語形式】

long readdir(FS_DIR* *dir*, FS_STAT* *dirstat*);

【パラメータ】

I/O	パラメータ	説明
I	FS_DIR* *dir*	状態を保持する FS_DIR 構造体を指定します。
O	FS_STAT* *dirstat*	ディレクトリ情報を格納する FS_STAT 構造体を指定します。

【機能】

- readdir() はディレクトリ情報構造体 *dirstat* へディレクトリエントリ情報を格納します。
- FAT のカレントディレクトリと親ディレクトリを指す、“.” と “..” も返されます。
- readdir() の呼び出しを繰り返すと、opendir() で指定したディレクトリ内のディレクトリあるいはファイルの情報を次々取得することができます。

【戻り値】

FS_OK:	**正常終了。**

FS_ENOTOPEN: オープンされていません。
FS_EPARAM: *dir* あるいは *dirstat* が NULL です。
FS_ENOENT: ディレクトリが存在しません。
FS_ENOTREADY: ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.12. closedir

【概要】

ディレクトリを閉じます。

【C 言語形式】

long closedir(FS_DIR* *dir*);

【パラメータ】

I/O	パラメータ	説明
I	FS_DIR* *dir*	状態を保持する FS_DIR 構造体を指定します。

【機能】

- closedir() はディレクトリ情報をクローズします。
- クローズ後は readdir() がエラー(FS_ENOTOPEN)になります。

【戻り値】

FS_OK: 正常終了。
FS_ ENOTOPEN: オープンされていません。

4.3.13. mkdir

【概要】

ディレクトリを作成します。

【C 言語形式】

long mkdir(const char* *path*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *pathname*	作成するディレクトリ名を指定します。

【機能】

- mkdir() は *pathname* で示される名前のディレクトリを作成します。

・ *pathname* はフルパスで指定します。

【戻り値】

FS_OK:	正常終了。
FS_ENODEV:	*pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。
FS_EACCESS:	親ディレクトリへの書き込み許可がありません。またはドライブが書き込み禁止です。
FS_EEXIST:	*pathname* が既に存在しています（それはディレクトリであるとは限りません）。
FS_ENAMETOOLONG:	*pathname* が長すぎます。
FS_ENODIR:	*pathname* の構成要素のディレクトリが存在しません。
FS_ENOSPC:	ドライブに十分な空きがありません。
FS_EPARAM:	*pathname* が NULL です。
FS_EINVAL:	*pathname* に使用できない文字が含まれています。
FS_ENOTREADY:	ドライブが準備されていません（媒体が装てんされていないなど）。

4.3.14. rmdir

【概要】

ディレクトリを削除します。

【C 言語形式】

long rmdir(const char* *pathname*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *pathname*	削除するディレクトリ名を指定します。

【機能】

・ rmdir() は *pathname* で示される名前のディレクトリを削除します。
・ *pathname* はフルパスで指定します。

【戻り値】

FS_OK:	正常終了。
FS_ENODEV:	*pathname* に指定されたドライブが見つかりません。あるいはドライブを指定していません。

FS_EACCESS: *pathname* のディレクトリあるいは親ディレクトリへの書き込み許可がありません。またはドライブが書き込み禁止です。

FS_ENAMETOLONG: *pathname* が長すぎます。

FS_ENOENT: *pathname* の構成要素のディレクトリが存在しません。

FS_ENOTEMPTY: ディレクトリに、“.”と“..”以外のエントリが存在します(ディレクトリが空ではありません)。

FS_ENOTREADY: ドライブが準備されていません(媒体が装てんされていないなど)。

4.3.15. mountfs

【概要】

ファイル・システムを使用できるようにします。

【C 言語形式】

long mountfs(FS_DRIVE* *drive* , const char* *name*);

【パラメータ】

I/O	パラメータ	説明
I	FS_DRIVE* *drive*	マウントするデバイスドライバの情報を格納した FSDRIVE 構造体を指定します。
I	const char* *drivename*	ドライブ名を指定します。

【機能】

- mountfs() はデバイスドライバを *drivename* で示されたドライブ名でファイル・システムにマウントします。ファイル・システムを使用する際は、最初にマウントする必要があります。マウントすることにより、デバイスドライバの関数へのアドレステーブルが、ファイル・システムに登録されることになり、ファイル・システムはデバイスドライバを呼び出すことが可能になります。
- *drive* に指定する構造体は、デバイスドライバごとに用意されています。
- ファイル・システムに複数の異なるデバイスドライバをマウントすることができます。

【戻り値】

FS_OK: 正常終了。

FS_EALREADY: 既にそのドライブ名でマウントされています。

FS_ENAMETOLONG: *drivename* が長すぎます。

FS_EPARAM: *drive* あるいは *drivename* が NULL です。

FS_ENOTREADY: ドライブが準備されていません(媒体が装てんされていないなど)。

FS_EINVAL: *drivename* に使用できない文字が含まれています。
FS_EFAIL: ドライブが FAT ファイル・システムでない、または初期化されていないことを示します。必要に応じて FAT 形式で初期化してください。

4.3.16. umountfs

【概要】

マウントしたファイル・システムを開放します。

【C 言語形式】

long umountfs(const char* *drive*);

【パラメータ】

I/O	パラメータ	説明
I	const char* *drivename*	mountfs()で登録済みのドライブ名を指定します。

【説明】

・ umountfs() はデバイスドライバをファイル・システムからアンマウントします。アンマウントした後は、そのドライブへのアクセスはできなくなります。

【戻り値】

FS_OK: 正常終了。
FS_ENOENT: マウントされていません。
FS_EPARAM: *drivename* が NULL です。

4.4. 構造体

4.4.1. FS_FILE 構造体

オープンしているファイルの情報を保持する構造体です。

構造体を宣言しているヘッダファイル：fs.h

4.4.2. FS_STAT 構造体

stat() あるいは readdir() で取得したディレクトリエントリの情報を格納する構造体です。

構造体を宣言しているヘッダファイル：fs.h

4.4.3. FS_DIR 構造体

オープンしているディレクトリの情報を保持する構造体です。

構造体を宣言しているヘッダファイル：fs.h

4.4.4. FS_DRIVE 構造体

ファイル・システムにデバイスドライバを登録する際に使用します。ドライブ名をドライバの各関数へのポインタを格納してファイル・システムに登録することにより、”/<ドライブ名>”でのアクセスが可能になります。

構造体を宣言しているヘッダファイル：diskio.h