FreeBSDの
デバイスドライバについて

デバイスドライバ

- 多くのオペレーティングシステムでは、入出力デバイスへのアクセスはデバイスドライバと呼ばれるデバイス依存の処理を行なうインターフェイス部分を経由して行なう

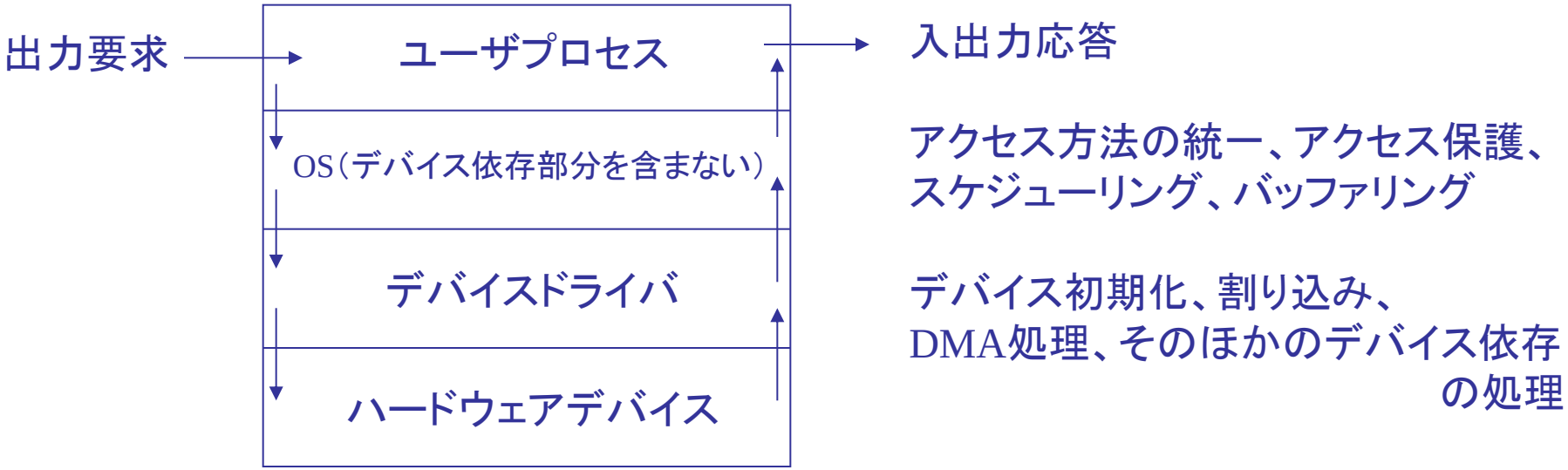
- 各種デバイスを簡単に利用できるようにする
  - 各種の入出力デバイスの詳細を，ユーザは意識しない
  - デバイスとしては性質の異なるディスクもフロッピーディスクも同様の方法でマウントすることができ、ファイルの入出力を行なうことができる
- リソースマネージャとしての機能
  - 複数のプロセスが同時に動作している状態でも，ユーザは安全にデバイスを利用することができる。
  - 例えば，複数のプロセスから同時にディスクに対するアクセスを行なっても，ファイルの内容が混ざったり、命令の混合によってデバイスが誤動作を起こしたりしない。

キャラクタ型、ブロック型

- UNIX系のOSの場合、キャラクタ型デバイスとブロック型デバイスに大別できる
- キャラクタ型デバイス:
  - デバイス特有の性質に応じたデータ長単位の入出力を行なう
  - シリアル、キーボード、パラレルインターフェイスなど標準的なデバイスの多くでは、1バイト単位の入出力を行ないます
- ブロック型デバイス:
  - ある大きさのバッファをもち、バッファリングが行なわれるのが特徴です
- 一般的には，ブロック型のデバイスも，キャラクタ型としての利用も可能

エントリポイント

- デバイスドライバには利用するためのエントリポイント(関数)が必要
  - 主なものは open, close, read, write, ioctl など
  - エントリポイントは，ユーザプログラムがデバイスへのアクセスを行なうときに利用される
  - カーネルにデバイスを登録するときに使用するデバイス構造体(cdevsw)に記述される



デバイスを識別するための番号

- メジャー番号(8ビット) とマイナー番号(24ビット)の組
  - メジャー番号： 個々のデバイスドライバに対応．最大256種類のデバイスドライバが存在可能です
  - マイナー番号： 同じドライバで扱われる機器の番号に対応．同じデバイスの何台目かを識別したり、デバイスの機能を指定するために用いられます

デバイスドライバに対する操作

- デバイスの初期化
- open/close/read/write
- Ioctl

など

デバイスの初期化

- プロセスの入出力要求の処理には，「デバイスの初期化」が必要
  - ファイルシステムのモジュールから，各デバイスドライバが「ファイル」として見える
  - 「デバイスの初期化」によって，デバイスドライバが物理的な入出力装置の存在を確認し、必要な初期化を行なって，動作可能になる
  - 初期化の過程では、登録された各デバイスドライバにつぎの2つの操作が行われる

1. プローブ
   ・入出力装置の存在の確認
2. アタッチ
   ・入出力装置を扱えるようにデバイスドライバ内のデータ構造を設定
   ・割り込み処理ルーチンを登録する

デバイスの初期化に必要な情報

- デバイスドライバでこれらの操作を行なうためには最低でも次の2つが既知である必要がある
  - I/Oアドレス
    入出力装置の制御ハードウェアとデータ交換するためにI/O命令で使用するアドレス
  - 割り込み番号
    入出力装置の状態が変化したときにCPUへ通知する

open/close/read/write

- デバイスドライバを利用したデータの入出力
  - 一般のファイル同様に，デバイスファイルの open, read, write で行う
- デバイスファイルの役割
  - ユーザプログラムがopen/close、read/writeを行なうためのエントリとして用意されているのが/dev以下に置かれているデバイスファイル
  - デバイスファイルをopenすることでデバイスドライバへの入出力の準備を行なう
  - デバイスファイルへのread/writeを行なうことで、デバイスドライバへのread/writeファンクションを実行する

ioctl

- デバイスドライバ特有の機能
- デバイスやデバイスドライバの動作モードの設定などを行なうためのもの
- read/writeの一般的な入出力機能ではできないことをするためのインターフェイスです

マウス用ドライバ(psm.c)

以下の構造体は、カーネルにマウスを登録するときに使用するデバイス構造体 (cdevsw) の例

```
static struct cdevsw psm_cdevsw = {
        /* open */      psmopen,
        /* close */     psmclose,
        /* read */      psmread,
        /* write */     nowrite,
        /* ioctl */     psmioctl,
        /* poll */      psmpoll,
        /* mmap */      nommap,
```

```
        /* strategy */      nostrategy,
        /* name */          "psm",
        /* maj */CDEV_MAJOR, /* #define CDEV_MAJOR 21 */
        /* dump */          nodump,
        /* psize */         nopsize,
        /* flags */         0,
        /* bmaj */          -1
};
```

以下，マウス用のドライバの基本的な部分であるpsmopen, psmclose, psmread,psmioctl　の説明を行う

static d_open_t psmopen(dev_t dev, int flag, int fmt, struct proc *p)

- デバイスdevのオープン処理
- 返り値:
  - 正常に処理が終了: 0
  - エラー: 整数値
- 引数
  - flag: システムコールopen()の第二引数
  - fmt: 文字型であることを示すS¥_IFCHR
  - p: オープンを要求したプロセス
- psmopen 関数は，システムコールopen()のたびに呼ばれる

static d_close_t psmclose(dev_t dev, int flag, int fmt, struct proc *p)

- デバイスdevのクローズ処理
- 返り値
  - 正常に処理が終了: 0
  - エラー: 整数値
- 引数
  - flag: ファイル記述子に設定されたフラグ
  - fmt: 文字型であることを示すS¥_IFCHR
  - p: オープンを要求したプロセス
- psmclose 関数は，devを参照するvノード(ファイル/ディレクトリに関する固定長の管理情報(iノード)を抽象化したもの)がなくなるときに呼ばれる
（システムコールclose()が呼ばれるたびに毎回呼ばれるとは限らない）

static d_read_t psmread(dev_t dev, struct uio *uio, int flag)

- デバイスdevからuioで表わされる領域へデータを読み出す
- 返り値
  - 正常に処理が終了 : 0
  - エラー: 正整数値
- 引数
  - flag: IO¥_NDELAYが指定される場合があるそうです
- psmread 関数は，システムコールread()のたびに毎回呼ばれる

static d_ioctl_t psmioctl(dev_t dev, u_long cmd, caddr_t addr, int flag, struct proc *p)

- システムコールioctl()の処理のために呼ばれる
  - デバイスdevに対して，cmdとaddrで表わされる制御コマンドを適用する
- 返り値
  - 正常に処理が終了： 0
  - エラー： 正整数値
- 引数
  - flag： ファイル記述子に設定されたフラグ
- システムコールioctl()の引数は，カーネルのメモリ空間にコピーされ、戻るときに必要に応じてプロセスのメモリ空間にコピーされる

マウスの状態を表わす構造体

```
struct psm_softc {                      /* Driver status information */
    struct selinfo rsel;        /* Process selecting for Input */
    unsigned char state;        /* Mouse driver state */
    int         config;         /* driver configuration flags */
    int         flags; /* other flags */
    KBDC        kbdc;                   /* handle to access the keyboard controller */
    struct resource *intr;      /* IRQ resource */
    void    *ih;                /* interrupt handle */
    mousehw_t   hw;                     /* hardware information */
    mousemode_t mode;                   /* operation mode */
    mousemode_t dflt_mode;      /* default operation mode */
```

```
    mousestatus_t status;     /* accumulated mouse movement */
    ringbuf_t     queue;      /* mouse status queue */
    unsigned char ipacket[16];        /* interim input buffer */
    int           inputbytes;     /* # of bytes in the input buffer */
    int           button;         /* the latest button state */
    int                   xold;   /* previous absolute X position */
    int                   yold;   /* previous absolute Y position */
    int                   watchdog;       /* watchdog timer flag */
    struct callout_handle callout;      /* watchdog timer call out */
    dev_t    dev;
    dev_t    bdev;
};
```

文字型の入出力の引数に使われる構造体

struct uio(/usr/include/sys/uio.h)

```
struct uio {
  struct iovec *uio_iov;     /* 領域の「先頭アドレスとバイトサイズ」 */
  int          uio_iovcnt;    /* 領域の個数 */
  off_t        uio_offset;    /* データを詰めた/取り出したアドレス */
  int          uio_resid;     /* 入出力するデータ数 */
  enum     uio_seg uio_segflg; /* 領域のアドレス空間(プロセス/カーネル) */
  enum     uio_rw uio_rw;     /* 入力、出力の区別 */
  struct proc *uio_procp;    /* 使用しているプロセス */
};
```

・デバイスドライバでは uiomove(caddr_t cp, int n, struct uio *uio)を用いて、cp から uio へ n バイトをコピー(入力の場合)、あるいは、uio からcp へ n バイトをコピー(出力の場合)を行なう

・正しくコピーできれば0が、さもなければエラーを表わす正整数が返される

・コピーの方向はuioに設定された情報をuiomove()が判断して決定され、uioのメンバーも適宜更新される。

```
static int
psmopen(dev_t dev, int flag, int fmt, struct proc *p)
{
/* マウスの状態を確認し、問題があればエラーを返す。問題がなければ、マウ
   スの状態を初期化し、マウスがオープン状態であることを記録する。*/
    int unit = PSM_UNIT(dev);
    struct psm_softc *sc;
    int command_byte;
    int err;
    int s;
    /* Get device data */
    sc = PSM_SOFTC(unit);
    if ((sc == NULL) || (sc->state & PSM_VALID) == 0)
        return (ENXIO); /* マウスの状態を確認して問題があればENXIOを返す */
    if (sc->state & PSM_OPEN)
        return (EBUSY); /* デバイスが既にオープンされているならばEBUSYを返す */
```

```
device_busy(devclass_get_device(psm_devclass, unit));

/* マウスの状態を初期化 */
sc->rsel.si_flags = 0;
sc->rsel.si_pid = 0;
sc->mode.level = sc->dflt_mode.level;
sc->mode.protocol = sc->dflt_mode.protocol;
sc->watchdog = FALSE;

/* マウスのイベントキューをクリア */
sc->queue.count = 0;
sc->queue.head = 0;
sc->queue.tail = 0;
sc->status.flags = 0;
```

```
sc->status.button = 0;
sc->status.obutton = 0;
sc->status.dx = 0;
sc->status.dy = 0;
sc->status.dz = 0;
sc->button = 0;

/* 入力バッファをクリア */
bzero(sc->ipacket, sizeof(sc->ipacket));
sc->inputbytes = 0;

/* まだいろいろと処理を行なうが長いので省略 */

err = doopen(unit, command_byte);
```

```
    /* done */
    if (err == 0)
        sc->state |= PSM_OPEN;
    kbdc_lock(sc->kbdc, FALSE);
    return (err);
}

static int
psmread(dev_t dev, struct uio *uio, int flag)
{
 /* マウスの状態を調べて、問題があればエラーを返す。問題がなければ、マウスから(マウスが使用しているバッファから)uioにデータを読み出す */
    register struct psm_softc *sc = PSM_SOFTC(PSM_UNIT(dev));
```

```
unsigned char buf[PSM_SMALLBUFSIZE];
 int error = 0;
 int s;
 int l;
 /* マウスに問題があればEIOを返す */
 if ((sc->state & PSM_VALID) == 0)
         return EIO;

 /* block until mouse activity occured */
 s = spltty();
 while (sc->queue.count <= 0) {
    if (PSM_NBLOCKIO(dev)) {
        splx(s);
        return EWOULDBLOCK;
    }
```

```
    sc->state |= PSM_ASLP;
    error = tsleep((caddr_t) sc, PZERO | PCATCH, "psmrea", 0);
    sc->state &= ~PSM_ASLP;
    if (error) {
        splx(s);
        return error;
    } else if ((sc->state & PSM_VALID) == 0) {
        /* the device disappeared! */
        splx(s);
        return EIO;
          }
}
splx(s);
```

```
/* copy data to the user land */
    while ((sc->queue.count > 0) && (uio->uio_resid > 0)) {
     /* 入力すべきデータがあるならば*/
        s = spltty();
            l = min(sc->queue.count, uio->uio_resid);
            if (l > sizeof(buf))    /* sizeof(buf)はバッファの最大値 */
                l = sizeof(buf);
     /* キューに入っているデータをバッファにコピー */
            if (l > sizeof(sc->queue.buf) - sc->queue.head) {
                bcopy(&sc->queue.buf[sc->queue.head], &buf[0],
                        sizeof(sc->queue.buf) - sc->queue.head);
                bcopy(&sc->queue.buf[0],
                        &buf[sizeof(sc->queue.buf) - sc->queue.head],
                        l - (sizeof(sc->queue.buf) - sc->queue.head));
```

```
            } else {
                bcopy(&sc->queue.buf[sc->queue.head], &buf[0], l);
            }
    /* キューのカウントを減らし、キューの先頭を移動 */
            sc->queue.count -= l;
            sc->queue.head = (sc->queue.head + l) % sizeof(sc->queue.buf);
        splx(s);
    /* 入力として受け取った構造体 uio にデータをコピー */
        error = uiomove(buf, l, uio);
        if (error)
                break;
    }
    return error;
}
```

```
static int
psmioctl(dev_t dev, u_long cmd, caddr_t addr, int flag, struct proc *p)
{
  /* マウスに対する様々な操作(マウスの情報を得る、マウスの設定をするなど)を行なう。cmdで行なうコマンドを指定。*/
    struct psm_softc *sc = PSM_SOFTC(PSM_UNIT(dev));
    mousemode_t mode;
    mousestatus_t status;
#if (defined(MOUSE_GETVARS))
    mousevar_t *var;
#endif
    mousedata_t *data;
```

```
int stat[3];
    int command_byte;
    int error = 0;
    int s;

    /* Perform IOCTL command */
    switch (cmd) {

    case OLD_MOUSE_GETHWINFO:
            s = spltty();
        ((old_mousehw_t *)addr)->buttons = sc->hw.buttons;
        ((old_mousehw_t *)addr)->iftype = sc->hw.iftype;
        ((old_mousehw_t *)addr)->type = sc->hw.type;
        ((old_mousehw_t *)addr)->hwid = sc->hw.hwid & 0x00ff;
```

```
            splx(s);
        break;
    case MOUSE_GETHWINFO:
        /* … 長いので省略 … */
        break;
    case OLD_MOUSE_GETMODE:
        /* … 長いので省略 … */
        break;
    case MOUSE_GETMODE:
        break;
    case OLD_MOUSE_SETMODE:
    case MOUSE_SETMODE:
        /* … 長いので省略 … */
        break;
```

```
case MOUSE_GETLEVEL:
        /* … 長いので省略 … */
        break;
    case MOUSE_SETLEVEL:
        /* … 長いので省略 … */
        break;
    case MOUSE_GETSTATUS:
        /* … 長いので省略 … */
        break;
#if (defined(MOUSE_GETVARS))
    case MOUSE_GETVARS:
        /* … 長いので省略 … */
        break;
    case MOUSE_SETVARS:
```

```
            return ENODEV;
#endif /* MOUSE_GETVARS */
    case MOUSE_READSTATE:
    case MOUSE_READDATA:
        /* … 長いので省略 … */
            break;
#if (defined(MOUSE_SETRESOLUTION))
    case MOUSE_SETRESOLUTION:
        /* … 長いので省略 … */
        break;
#endif /* MOUSE_SETRESOLUTION */
#if (defined(MOUSE_SETRATE))
    case MOUSE_SETRATE:
        /* … 長いので省略 … */
        break;
```

```
#endif /* MOUSE_SETRATE */
#if (defined(MOUSE_SETSCALING))
    case MOUSE_SETSCALING:
        /* … 長いので省略 … */
        break;
#endif /* MOUSE_SETSCALING */
#if (defined(MOUSE_GETHWID))
    case MOUSE_GETHWID:
        /* … 長いので省略 … */
        break;
```

```
#endif /* MOUSE_GETHWID */
    default:
            return ENOTTY;
    }
    return error;
}
```