システムソフトウェア

2020年度

第12回 (11/16)

月曜7-8限・木曜7-8限(Zoom)

講義担当：渡部卓雄 (Takuo Watanabe)

http://titech-os.github.io

e-mail: takuo∅c.titech.ac.jp

本日のメニュー

- I/Oシステム

I/O

- 外部機器（I/O機器）との通信（を行う部分）
- 外部機器
  - キーボード，マウス
  - ディスプレイ
  - タイマ，RTC
  - 汎用ポート
    - シリアルポート，パラレルポート
    - USB

接続方式

- メモリマップトI/O
  - 外部機器がメモリバスに接続され，外部機器はメモリの一部としてアクセスされる方式
    - I/Oのための命令は不要
    - メモリ空間を圧迫する
- ポートマップトI/O（I/Oマップト I/O）
  - 外部機器が（メモリバスとは別の）専用のインターフェース（I/Oポート）に接続される方式
    - I/Oポートの機構とI/Oのための命令が必要
    - メモリ空間を圧迫しない

I/Oコントローラ

- 実際の機器（キーボード，etc.）とCPUを結ぶインターフェースとなる回路
- ハードウェアインターフェイス
  - アドレス（I/Oポートあるいはメモリ）
  - データ
  - 割り込み信号線
- ソフトウェアインターフェース
  - 状態(status)レジスタ
  - データレジスタ

I/Oの方式

- I/Oコントローラのデータレジスタをアクセスすることで入出力を行う
  - いつでもできるわけではない
  - 多くの場合，状態レジスタの特定のビット（フラグ）が決められた値をもつときのみ，有効な入出力が行える.
- したがって，入出力を行う際は，まず状態レジスタを読んでフラグの値を調べてからデータのやり取りを行う

ポーリング(polling)

- 状態フラグの値を繰り返し検査する方式

```
while ((inb(dev_stat) & IN_OK) == 0);
data = inb(dev_data);
```

```
while ((inb(dev_stat) & OUT_OK) == 0);
outb(dev_data, data);
```

  - この例のinb/outbはx86のI/O命令
- CPUを占有するので待ち時間が長くなるような場合には使えない

割り込み

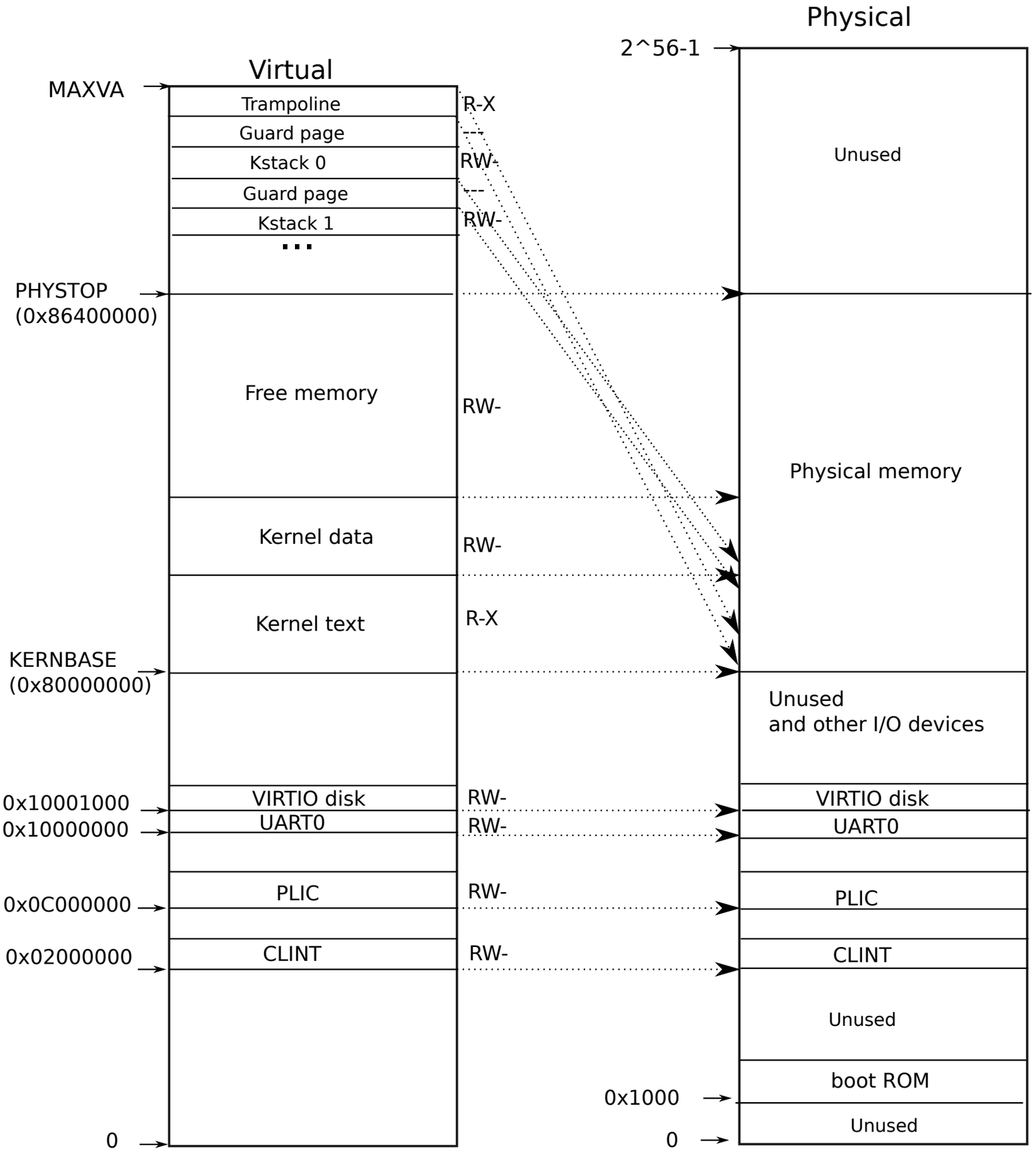
- I/Oコントローラによっては，所定の状態になったとき（フラグが所定の値になるとき）に，同時に割り込み信号線がアクティブになる．これによってCPUに割り込みが発生する
- I/O操作は割り込みハンドラを介して行う

DMA (Direct Memory Access)

- 外部機器がCPUを介さずに直接バスにアクセスしてメモリの読み書きを行うこと
- CPUによるロード・ストア（の繰り返し）によらず，高速にデータ転送ができる
  - HDD，グラフィクス，ネットワーク，USB等
- 転送方式
  - サイクルスチール
    - CPUのバスサイクル中で，バスへのアクセスを行なっていないときに転送を行う
  - バースト
    - バスを占有してまとまった量のデータ転送を行う

xv6のI/O

- x86版：ポートマップトI/O
  - IN/OUT命令を利用
- RISC-V版：メモリマップトI/O
  - UART
    - 仮想16550
  - ディスク
    - Virtioと呼ばれる仮想デバイス規格に沿っている
  - PLIC
  - CLINT

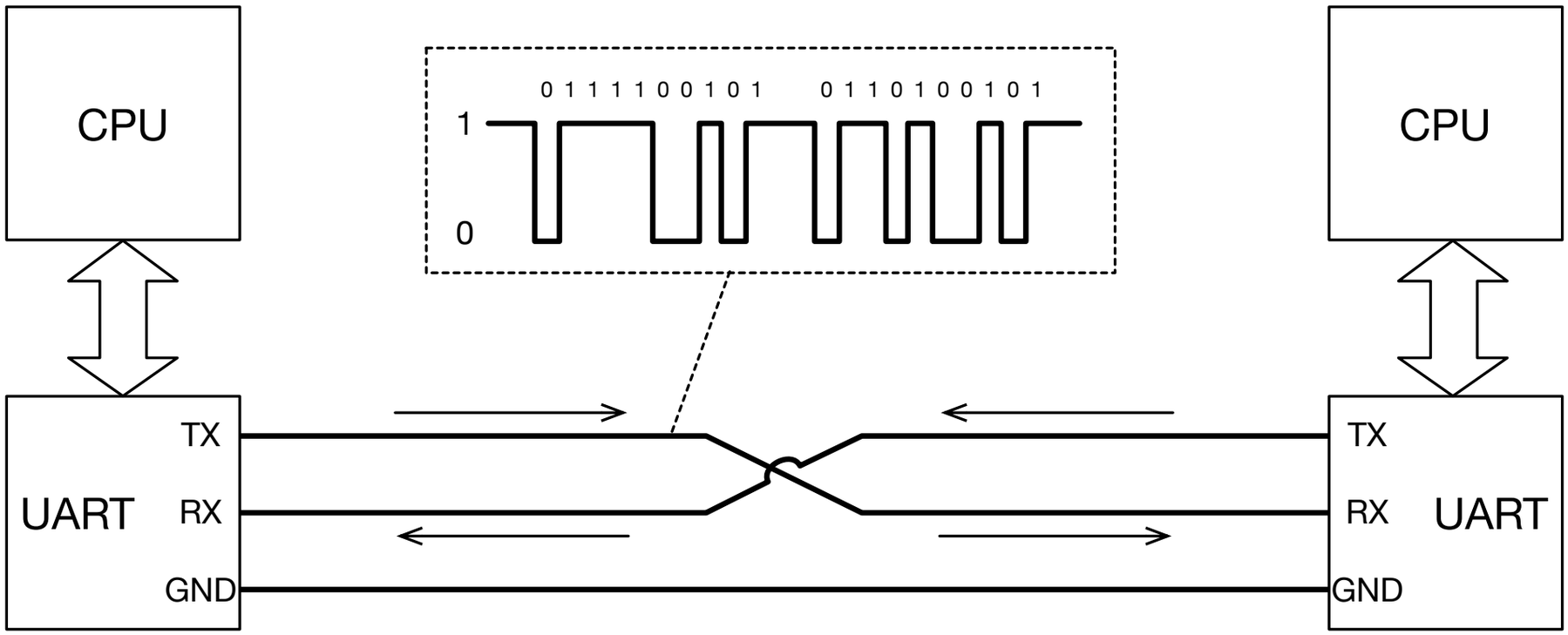

Physical
2^56-1
Virtual
MAXVA
Trampoline
R-X
Guard page

Kstack 0
RW-
Guard page

Kstack 1
RW-
...
Unused
PHYSTOP
(0x86400000)
Free memory
RW-
Physical memory
Kernel data
RW-
Kernel text
R-X
KERNBASE
(0x80000000)
Unused
and other I/O devices
0x10001000
VIRTIO disk
RW-
VIRTIO disk
0x10000000
UART0
RW-
UART0
0x0C000000
PLIC
RW-
PLIC
0x02000000
CLINT
RW-
CLINT
Unused
boot ROM
0x1000
Unused
0

UART

- 調歩同期式によるシリアル通信方式およびそのためのI/O機器
  - Universal Asynchronous Receiver/Transmitter



シリアルポート

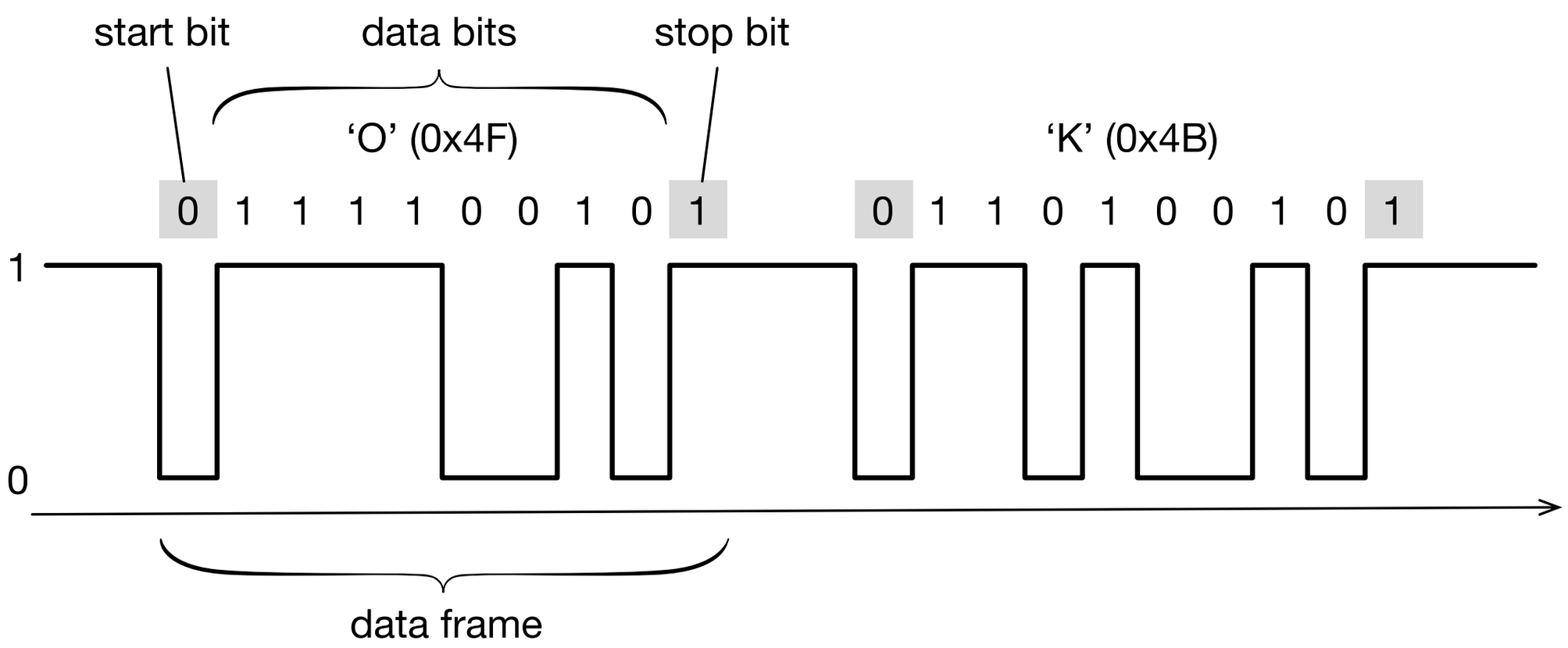
かつては多くのコンピュータにRS-232Cと呼ばれる規格のシリアルポートのインターフェースが備えられており（左の写真はRS-232Cのコネクタの一例），このインターフェースを介してコンピュータ本体と端末装置を接続していた.

端末装置はキーボードとディスプレイ（あるいはプリンタ）からなり，キーをタイプすると対応する文字コードがシリアル回線を経由してコンピュータに送信される．またコンピュータからの出力を受信すると文字として画面に表示する.

シリアル回線
(9600bps)

写真はWikipediaより

調歩同期式による通信



送信側は，スタートビット(0)に続けて8個のビットを一定の速度で順に送信し，最後にストップビット(1)を送信して1バイトの送信を終える．ストップビット送信後に次のスタートビットを送信するまでの時間は任意である．受信側は，0（スタートビット）を検出したのち，一定の速度で8個のビットを受信し，その後に1（ストップビット）を検出して1バイトの受信を終える．

通信速度

- bps (bit per second) で表す
  - ボー（baud）と混同しないこと
- 調歩同期式では送信側と受信側で速度を合わせる必要がある
- よく使われる速度(bps)
  - 110, 300, 600, 1200, 4800, 9600, 19200, 38400, 57600, 115200

xv6でのUART

- PC-AT互換機でよく用いられていた16550というUARTを模倣したもの
  - アドレス：UART0～UART0+7
    - UART0 = 0x10000000
    - 次頁の表のアドレス(A2A1A0)が011の場合は，UART0+3に割り当てられる
    - 各レジスタは8ビット

16550 UARTのレジスタ

アドレス			モード	
A2	A1	A0	読み出し	書き込み
0	0	0	受信データ(RHR)	送信データ(THR) / Divisor latch LSB
0	0	1	-	割り込み許可(IER) / Divisor latch MSB
0	1	0	割り込みステータス(ISR)	-
0	1	1	-	ライン制御(LCR)
1	0	0	-	モデム制御(MCR)
1	0	1	ラインステータス(LSR)	-
1	1	0	モデムステータス(MSR)	-
1	1	1	スクラッチレジスタ(SPR)	スクラッチレジスタ(SPR)

UARTの初期化

```
#define Reg(reg) ((volatile unsigned char *)(UART0 + reg))
#define RHR 0
#define THR 0
#define IER 1
#define FCR 2
#define ISR 2
#define LCR 3
#define LSR 5
#define ReadReg(reg) (*(Reg(reg))
#define WriteReg(reg, v) (*(Rge(reg)) = (v))

void uartinit() {
  WriteReg(IER, 0x00);  // 割り込みを無効化
  WriteReg(LCR, 0x80);  // 通信速度設定モード開始
  WriteReg(0, 0x03);    //     速度パラメータ下位8ビット
  WriteReg(1, 0x00);    //     速度パラメータ上位8ビット
  WriteReg(LCR, 0x03);  // 通信速度設定モード終了
                        // データ長8ビット，パリティなしに設定
  WriteReg(FCR, 0x07);  // 送受信FIFOを有効化し，FIFOをクリア
  WriteReg(IER, 0x01);  // 割り込みを有効化
}
```

1バイトの入出力

```
// 1文字出力
void uartputc(int c) {
  // 送信FIFOに空きが出るまで待つ
  while ((ReadReg(LSR) & (1 << 5)) == 0);
  // 1バイトをFIFOに書き込む
  WriteReg(THR, c);
}

int uartgetc() {
  // 受信FIFOにデータがあるかチェック
  if (ReadReg(LSR) & 0x01) {
    // あれば1バイト読み込む
    return ReadReg(RHR);
  else
    return -1;
}
```

コンソール入出力

- いわゆるターミナルとの入出力
  - 多くの場合はキーボード入力＋画面出力
  - xv6-riscvではUART入出力として実装
  - コンソール：多くのシステムでは管理用端末
- インターフェース
  - consoleread
    - コンソールから1文字入力する
    - **割り込みを用いた**入力の処理を行う
  - consolewrite
    - コンソールへ1文字出力する
    - 割り込みは用いずに直接出力する.

コンソールデバイス

```
// コンソール（入力バッファ）
struct {
  struct spinlock lock;
  char buf[INPUT_BUF];
  uint r;
  uint w;
  uint e;
} cons;

int consolewrite(int user_src, uint64 src, int n) { ... }
int consoleread(int user_dst, uint64 src, int n) { ... }


...

void consoleinit() {
  initlock(&cons.lock, "cons");
  uartinit();
  devsw[CONSOLE].read = consoleread;
  devsw[CONSOLE].write = consolewrite;
}
```

デバイスファイル

```
struct devsw {
  int (*read)(int, uint64, int);
  int (*write)(int, uint64, int);
};
```

```
struct devsw devsw[NDEV];
```

```
int fileread(struct file *f, uint64 addr, int n) {
  ...
  else if (f->type == FD_DEVICE) {  // デバイスファイル
    ...
    r = devsw[f->major].read(1, addr, n);
  }
  else if (f->type == FD_INODE) {  // 通常のファイル
    ilock(f->ip);
    if ((r = readi(f->ip, 1, addr, f->off, n) > 0)
      f->off += r;
    iunlock(f->ip);
  }
  ...
  return r;
}
```

コンソールデバイスファイルの作成

```
int main() {
  ...
  if (open("console", O_RDWR) < 0) {
    mknod("console", 1, 1);
    open("console", O_RDWR);
  }
  dup(0); // stdout  (fd = 1)
  dup(0); // stderr  (fd = 2)

  ...
}
```

- プログラムinitにおいて上記のようにデバイスファイルconsoleを作成している
  - mknodでコンソール（デバイス番号1）のデバイスファイルを作成
  - openでfd=0のファイルをオープン
  - 続く2回のdupでfd=1,2のファイルを作成

コンソール入力

```
int consoleread(int usr_dst, uint64 dst, int n) {
  uint target = n;
  ...
  acquire(&cons.lock);
  while (n > 0) {
    while (cons.r == cons.w) {   // バッファに未読文字がない
      ...
      sleep(&cons.r, &cons.lock);
    }
    c = cons.buf[cons.r++ % INPUT_BUF];  // 1文字取り出す
    ...
    --n;
    ...
  }
  release(&cons.lock);
  return target - n;
}
```

- バッファから1文字入力する．入力すべき文字がない場合はsleepする．

デバイス割り込み

```
int devintr() {
  uint64 scause = r_scause();
  if ((scause & 0x8000000000000000L) && (scause & 0xff) == 9) {
    int irq = plic_claim();
    if (irq == UART0_IRQ)          // UART割り込み
      uartintr();
    else if (irq == VIRTIO0_IRQ)  // ディスク割り込み
      virtio_disk_intr();
    plic_complete(irq);
    return 1;
  }
  else if (scause == 0x8000000000000001L) {
    ...  // タイマー割り込みの処理（省略）
    return 2;
  }
  else return 0;
}
```

UART割り込み

```
void uartintr() {
  while (1) {
    int c = uartgetc();
    if (c == -1) break;
    consoleintr(c);
  }
}
```

- consoleintrは読み込んだ1文字(c)をバッファ(cons.buf)に追加する
  - その際，ctrl-H/DEL（1文字削除）やctrl-U（1行削除）などの処理を行う
  - 改行やEOF(ctrl-D)が入力された場合は，consolereadを起こす

コンソール入力（割り込み）

```
void consoleintr(int c) {
  acquire(&cons.lock);
  switch (c) {
  ...  // ctrl-P, ctrl-U, ctrl-H などの処理
  default:
    if (c != 0 && cons.e - cons.r < INPUT_BUF) {
      c = (c == '\r') ? '\n' : c;
      consputc(c);
      cons.buf[cons.e++ % INPUT_BUF] = c;
      if (c == '\n' || c == C('D') || cons.e == cons.r + INPUT_BUF) {
        cons.w = cons.e;
        wakeup(&cons.r);
      }
    }
    break;
  }
  release(&cons.lock);
}
```

コンソール出力

```
void consputc(int c) {
  ...
  if (c == BACKSPACE) {
    uartputc('\b'); uartputc(' '); uartputc('\b');
  }
  else
    uartputc(c);
}

void consolewrite(int user_src, uint64 src, int n) {
  int i;
  acquire(&cons.lock);
  for (i = 0; i < n; i++) {
    char c;
    if (either_copyin(&c, user_src, src + i, 1) == -1)
      break;
    consputc(c);
  }
  release(&cons.lock);
  return n;
}
```

xv6のファイルシステム階層

- システムコール(sysfile.c)
- ファイルディスクリプタ(file.c)
- パス名(fs.c)
- ディレクトリ(fs.c)
- Inode (fs.c)
- ログ(log.c)
- バッファキャッシュ(bio.c)
- 低レベルI/O (virtio_disk.c)

ディスク入出力

```
int writei(struct inode *ip, int user_src,
           uint64 src, uint off, uint n) {
  ...
  for (tot = 0; tot < n; tot += m, off += m, src += m) {
    bp = bread(ip->dev, bmap(ip, off / BSIZE));
    m = min(n - tot, BSIZE - off % BSIZE);
    if (either_copyin(bp->data + (off % BSIZE),
                      user_src, src, m) == -1) {
      brelse(bp);
      break;
    }
    log_write(bp);
    brelse(bp);
  }
  ...
  return n;
}
```

- breadで確保したバッファキャッシュに対する読み書きとして実現している

バッファキャッシュの読み書き

- **struct** buf *bread(uint dev, uint blockno)
  - ディスク上のブロックを指定し，その内容を反映したバッファキャッシュを確保
- **void** bwrite(**struct** buf *b)
  - 指定したバッファの内容をディスクに書き出す

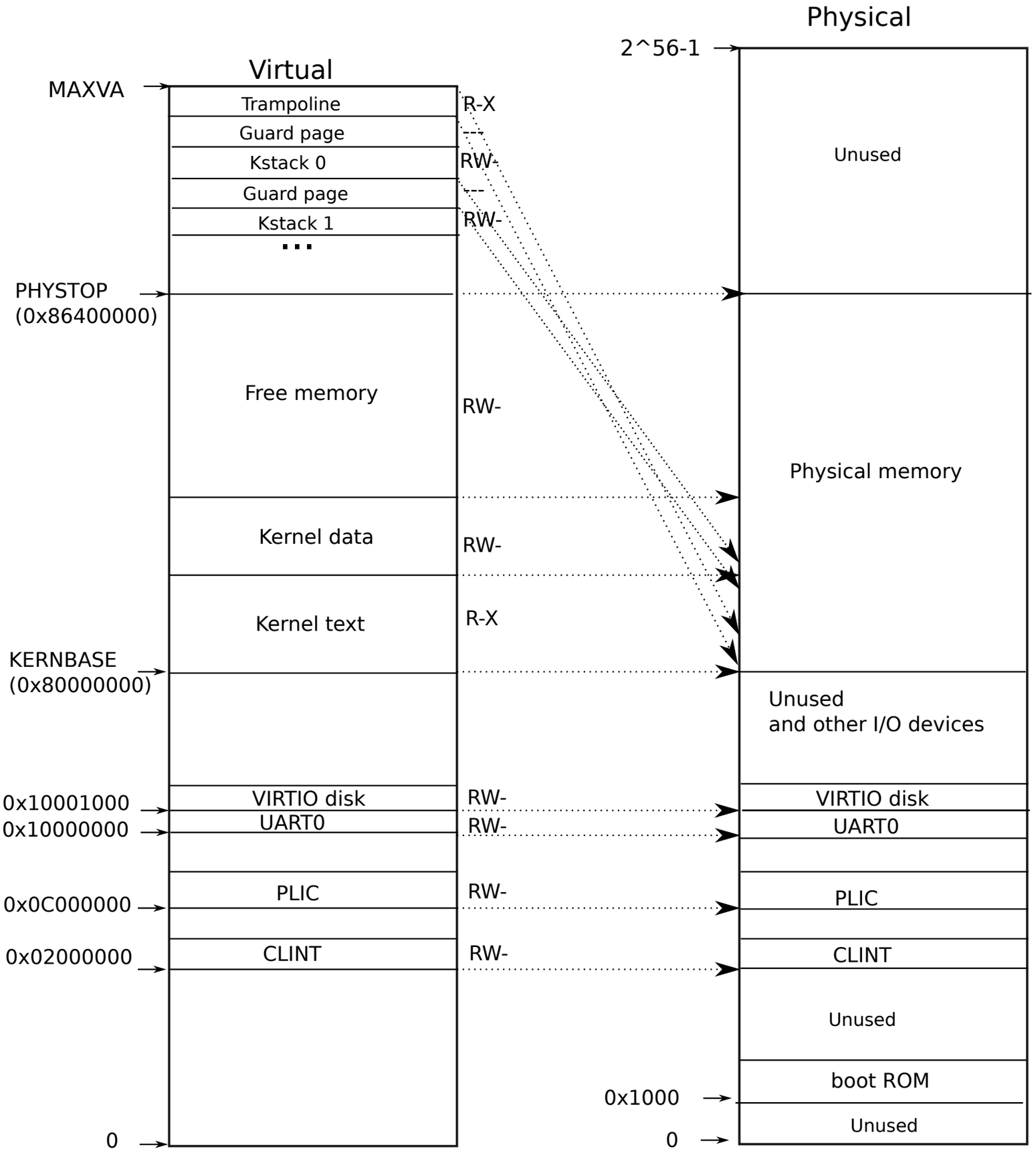
ディスク入出力

```
struct buf *bread(uint dev, uint blockno) {
  struct buf *b;
  b = bget(dev, blockno);
  if (!b->valid) {
    virtio_disk_rw(b, 0);
    b->valid = 1;
  }
  return b;
}

void bwrite(struct buf *b) {
  if (!holdingsleep(&b->lock))
    panic("bwrite");
  virtio_disk_rw(b, 1);
}
```

VirtIO

- Qemuによって模倣されるハードウェア
  - qemuのオプション -machine virt で指定
  - xv6ではUARTとディスクを用いている
- ドキュメント：xv6-riscv/doc
- ソース
  - https://github.com/qemu/qemu/blob/master/hw/riscv/virt.c


Physical
2^56-1
Virtual
MAXVA
Trampoline
R-X
Guard page

Kstack 0
RW-
Guard page

Kstack 1
RW-
...
Unused
PHYSTOP
(0x86400000)
Free memory
RW-
Physical memory
Kernel data
RW-
Kernel text
R-X
KERNBASE
(0x80000000)
Unused
and other I/O devices
0x10001000
VIRTIO disk
RW-
VIRTIO disk
0x10000000
UART0
RW-
UART0
0x0C000000
PLIC
RW-
PLIC
0x02000000
CLINT
RW-
CLINT
Unused
boot ROM
0x1000
Unused
0
0

VirtIO Disk

- IDEやSCSI等のインターフェースを介さず，メモリバス(Virtio-MMIO)に接続されている
  - アドレス：VIRTIO0 (0x10001000)
- ブロックデバイス（ブロック単位で読み書き）
- qemuのオプション
  - -drive file=fs.img,if=none,format=raw,id=x0
    -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

VirtIO Diskの読み書き

- DMA（Direct Memory Access）を利用
- 読み出し（書き込み）を行いたいバッファキャッシュのアドレスとディスクのセクタを指定して割り込みのリクエストを出す
- ディスクコントローラは指定されたアドレスから1ブロック分の読み書きを完了したのち割り込みを発生させる

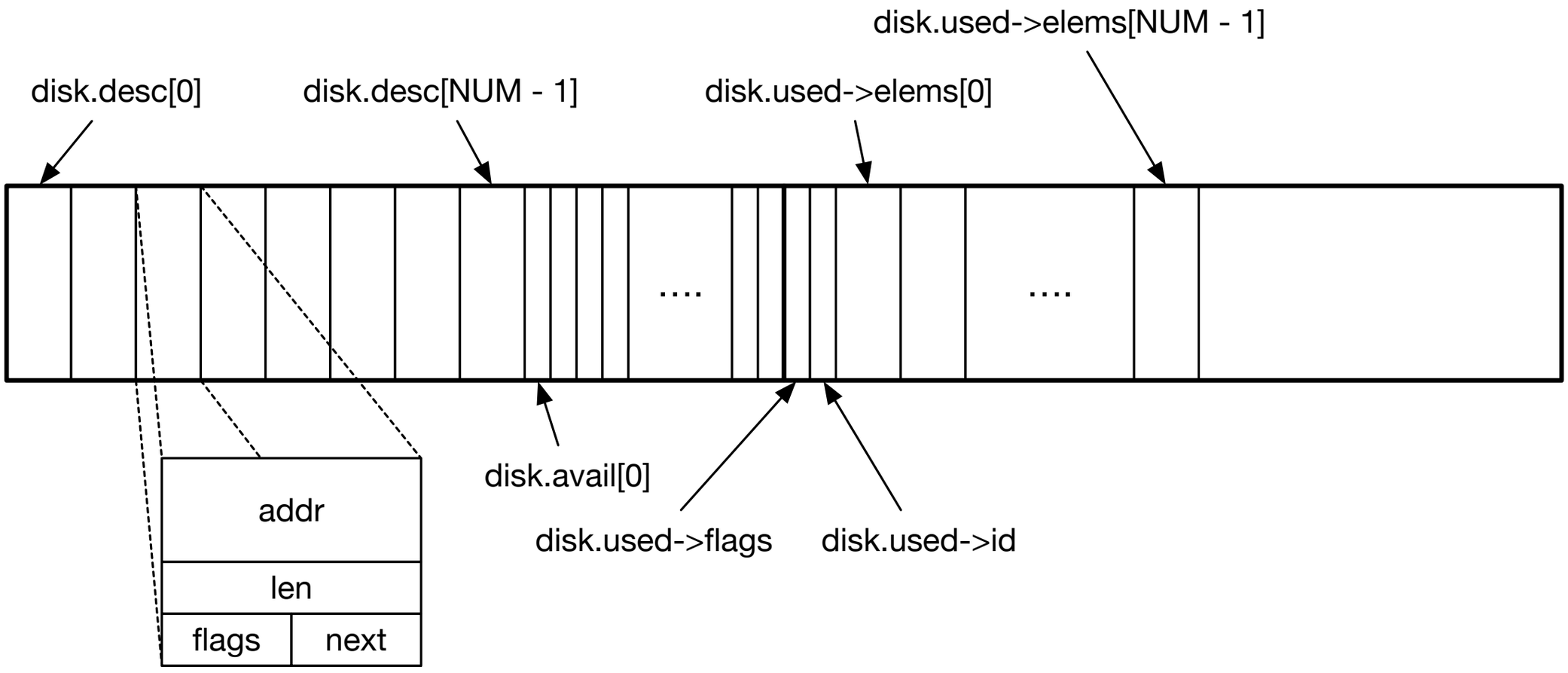
ディスク構造体

```
static struct disk {
  char pages[2 * PAGSIZE];  // コントローラに渡すパラメータの領域
  struct VRingDesc *desc;   // パラメータキュー
  uint16 *avail;            //
  struct UsedArea *used;    // コントローラからの返答

  char free[NUM];
  uint16 used_idx;
  struct {
    struct buf *b;
    char status;
  } info[NUM];

  struct spinlock vdisk_lock;
} __attribute__ ((aligned (PGSIZE))) disk;
```

disk.pages



- コントローラに渡すパラメタとコントローラからの返答を格納
  - パラメタは連結リストの形で作成され，最初の要素のインデックスがavail[2...]に渡される.

ディスク入出力の開始

```
void virtio_disk_rw(struct buf *b, int write) {

  acquire(&disk.vdisk_lock);

  // パラメタの設定

  // disk フィールドは現在転送中（転送をリクエストしてから完了する
  // までの間）であることを表す
  b->disk = 1;
  ...
  *R(VIRTIO_MMIO_QUEUE_NOTIFY) = 0;  // 転送のリクエスト

  while (b->disk == 1) {
    sleep(b, &disk.vdisk_lock);
  }
  ...
  release(&disk.vdisk_lock);
}
```

ディスクコントローラからの割り込み処理

```
void virtio_disk_intr() {
  acquire(&disk.vdisk_lock);

  while ((disk.used_idx % NUM) != (disk.used->id % NUM)) {
    int id = disk.used->elems[disk.used_idx].id;
    ...
    disk.info[id].b->disk = 0;
    wakeup(disk.info[id].b);

    disk.used_idx = (disk.used_idx + 1) % NUM;
  }

  release(&disk.vdisk_lock);
}
```

まとめ

- I/Oシステム
  - コンソールの入出力
  - UART
  - デバイスファイル
  - 割り込みによる入力
  - ディスクの低レベル入出力