システムソフトウェア

2021年度

第7回 (10/25)

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

- xv6のプロセス

xv6の起動

起動するカーネルのプログラム　　エミュレートするマシンの種類

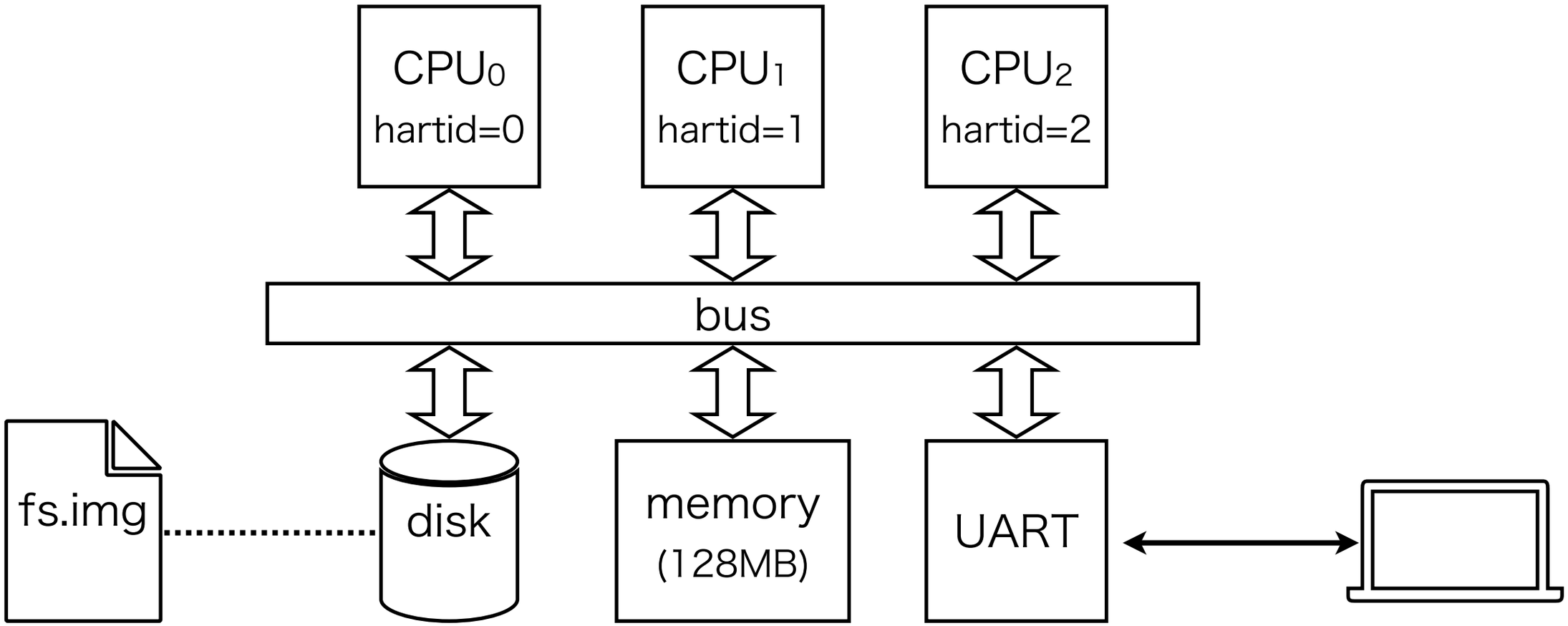
```
$ cd xv6-riscv
$ make qemu
qemu-system-riscv64 -machine virt -bios none \
    -kernel kernel/kernel -m 128M -smp 3 -nographic \
    -drive file=fs.img,if=none,format=raw,id=x0 \
    -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting
                                  メモリサイズ　　CPUの個数
hart 2 starting
hart 1 starting
init: starting sh
$
```

xv6-riscvの「ブート」

- 多くのコンピュータでは電源投入時（リセット時）に起動するプログラムがROMに入っている．そのプログラムは最初に自己診断を行い，続けて二次記憶装置に格納されているブートローダと呼ばれるプログラムをメモリにコピー（ロード）し，それに制御を移す.
- ブートローダはOSカーネルのコードを二次記憶装置からロードし，それに制御を移す.
  - 多くの場合，最初に起動するブートローダはより高機能なブートローダをロードする．最終的にOSカーネルをロードする前に複数のブートローダを経由する.
- 現在のxv6-riscvでは以上の手順は省略されており，最初から主記憶内にカーネルがロードされた状態で起動する.
  - QEMUの -kernel オプションでロードするファイルを指定している.

エミュレートされるマシン



kernel/kernel

```
$ file kernel/kernel
kernel/kernel: ELF 64-bit LSB executable, UCB RISC-V, version
1 (SYSV), statically linked, with debug_info, not stripped
$ rm kernel/kernel
$ make kernel/kernel
riscv64-unknown-elf-ld -z max-page-size=4096 \\
    -T kernel/kernel.ld -o kernel/kernel \\
    kernel/entry.o kernel/start.o kernel/console.o \\
    kernel/printf.o kernel/uart.o kernel/kalloc.o \\
    kernel/spinlock.o kernel/string.o kern
    l/vm.o kernel/proc.o kernel/swtch
    kernel/trampoline.o kernel/trap.o kernel/syscall.o \\
    kernel/sysproc.o kernel/bio.o kernel/fs.o kernel/log.o \\
    kernel/sleeplock.o kernel/file.o kernel/pipe.o \\
    kernel/exec.o kernel/sysfile.o kernel/kernelvec.o \\
    kernel/plic.o kernel/virtio_disk.o
riscv64-unknown-elf-objdump -S kernel/kernel > \\
    kernel/kernel.asm
riscv64-unknown-elf-objdump -t kernel/kernel | \\
    sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > \\
    kernel/kernel.sym
$
```

リンカスクリプト

コンパイルしたバイナリから
カーネルの実行形式を作成

デバッグ用
おまけ

kernel.ld（リンカスクリプト）

```
OUTPUT_ARCH( "riscv" )
ENTRY( _entry )

SECTIONS
{
  /*
   * ensure that entry.S / _entry is at 0x80000000,
   * where qemu's -kernel jumps.
   */
  . = 0x80000000;

  .text : {
    *(.text .text.*)
    . = ALIGN(0x1000);
    _trampoline = .;
    *(trampsec)
    . = ALIGN(0x1000);
    ASSERT(. - _trampoline == 0x1000, "error: trampoline
larger than one page");
    PROVIDE(etext = .);
  }
...
```

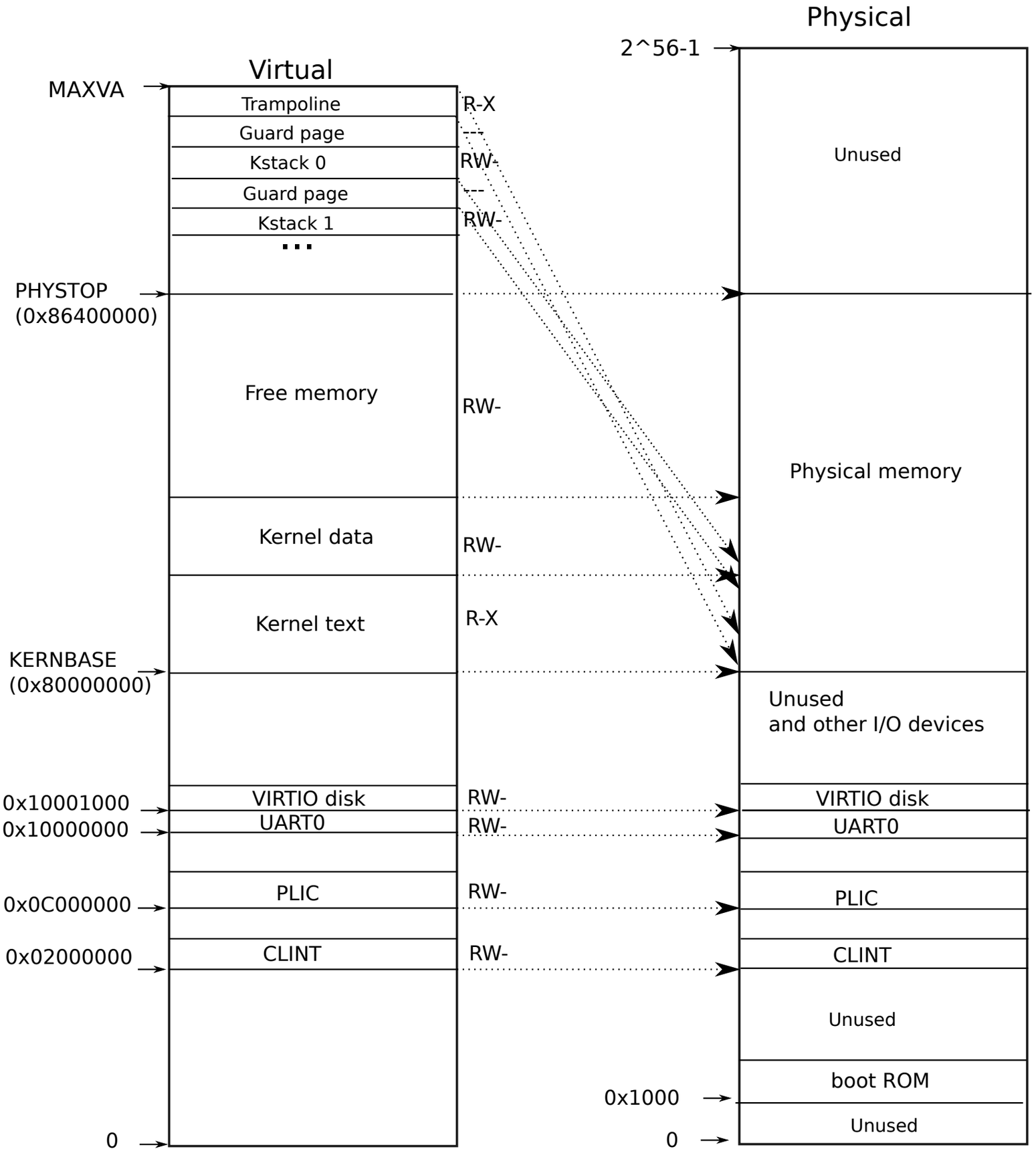
実行を開始する場所

カーネルの開始アドレス(KERNBASE)

テキストセクション（実行可能コード）の
直後にトランポリンと呼ばれる1ページの
セクション（これも実行可能コードの一部）を
作っている

メモリレイアウト

- memlayout.h
  - 主にカーネルのメモリレイアウトを決めるための定数（マクロ）の定義
- riscv.h
  - RISC-Vの特権命令を呼び出すためのインライン関数の定義
  - 各種フラグの読み書きのためのマクロ
  - 論理アドレスの最大値 MAXVA(=2^{38})


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

カーネルの起動(各CPU)

- 各CPUは _entry からスタートする.
- _entry (entry.S)
  - Cで書かれたプログラムが動作するようにスタックを設定して, start を呼び出す.
- start() (start.c)
  - 割り込み関連の初期化
  - タイマ割り込みの設定・有効化
  - マシンモードからスーパーバイザモードに移行するための準備
  - mret命令を実行してmainに制御を移す
    - この命令は本来マシンモードの割り込みから戻るときに用いられるが, そのときにより低いモード（スーパーバイザモードやユーザモード）に移行することができる．そこでmretの「戻り先」をmainの先頭アドレスに設定しておき，mainに「戻る」と同時にスーパーバイザーモードに移行する.

main (CPU0)

```
volatile static int started = 0;

// start() jumps here in supervisor mode on all CPUs.
void
main()
{
  if(cpuid() == 0){
    consoleinit();
    printfinit();
    printf("\n");
    printf("xv6 kernel is booting\n");
    printf("\n");
    kinit();         // physical page allocator
    kvminit();       // create kernel page table
    kvminithart();   // turn on paging
    procinit();      // process table
    trapinit();      // trap vectors
    trapinithart();  // install kernel trap vector
    plicinit();      // set up interrupt controller
    plicinithart();  // ask PLIC for device interrupts
    binit();         // buffer cache
    iinit();         // inode table
    fileinit();      // file table
    virtio_disk_init(); // emulated hard disk
    userinit();      // first user process
    __sync_synchronize();
    started = 1;
  } else {
    ...
  }

  scheduler();
}
```

とにかくコンソールに何か出せるようにしておく.

メモリ関連の設定

プロセステーブル初期化

割りこみ関連の設定

ファイルシステム関連

initの起動

main (CPU$_1$〜)

```
volatile static int started = 0;

// start() jumps here in supervisor mode on all CPUs.
void
main()
{
  if(cpuid() == 0){
    ...
    __sync_synchronize();
    started = 1;
  } else {
    while(started == 0)
      ;
    __sync_synchronize();
    printf("hart %d starting\n", cpuid());
    kvminithart();    // turn on paging
    trapinithart();   // install kernel trap vector
    plicinithart();   // ask PLIC for device interrupts
  }

  scheduler();
}
```

CPU$_0$を待つ

メモリ管理関連の設定

- kinit()　(kalloc.c)
  - kallocのための初期化処理
    - kallocはフリーメモリから1ページもらうための関数
    - 線形リストによる素朴なメモリ管理
- kvminit()　(vm.c)
  - カーネルのページテーブルを設定
- kvminithart()　(vm.c)
  - kvminitが作ったページテーブルを有効化

プロセスと割り込み関連の設定

- procinit (proc.c)
  - プロセステーブルの初期化
    - ロックの初期化とカーネルスタックの割り当て
- trapinit (trap.c)
  - 割り込み関連の初期化
    - タイマー割り込みのロックの初期化のみ
- trapinithart (trap.c)
  - カーネルモードでの割り込みベクタの設定

xv6-riscvでの割り込み

- ユーザモード
  - システムコール
  - タイマー割り込み
  - デバイス（UART, ディスク）からの割り込み
- スーパーバイザモード
  - タイマー割り込み
  - デバイス（UART, ディスク）からの割り込み

- タイマー割り込み
  - 一旦マシンモードに移行し，そこからスーパーバイザーモードの割り込みを発生させている

割り込みハンドラ(1)

- uservec (trampoline.S)
  - 論理アドレスの一番最上位にあるトランポリン(tranpoline)ページにあり，ユーザモードで発生した割り込みを処理する
  - レジスタ等をtrapframeに保存し，カーネルのページテーブルに切り替えてからusertrap()を呼ぶ
    - トランポリンページはカーネルの論理アドレスでも同じ位置にあるため，uservecはページテーブルを切り替えてからも問題なく動作する.

割り込みハンドラ(2)

- kernelvec (kernelvec.S)
  - スーパーバイザーモードで発生した割り込みを処理する.
  - レジスタをカーネルスタックに保存したのちkerneltrapを呼ぶ
- timervec (kernelvec.S)
  - タイマー割り込みのハンドラで，マシンモードで動作する
  - 次のタイマー割り込みを設定したのち，スーパーバイザーモードの割り込みを発生させる

スケジューラ

- scheduler (proc.c)
  - 各CPUが実行
  - プロセステーブル(proc)をみてモードが RUNNABLEになっているものがあったら RUNNINGに切り替え, swtchで制御を移す

プロセスの生成(fork)

- allocproc (proc.c)
  - プロセステーブル(proc)からUNUSEDなものをひとつ見つけて初期化
    - pidの割り当て
    - trapframeとページテーブルの設定
    - コンテクストの設定
      - カーネルスタックの底をコンテクストのspに設定
      - forkretをコンテクストの戻りアドレスに設定
- fork (proc.c)
  - allocprocでプロセスの「種」を生成
  - 親プロセスのメモリ空間をコピー
  - 親プロセスのレジスタ(trapframe)をコピー

プロセスの生成(init)

- userinit (proc.c)
  - allocprocでプロセスの「種」を作る
  - 1ページだけのメモリ空間を作り，initcodeの内容をコピーする
    - initcodeは exec("init") を実行するだけのコード
  - 当該ページの先頭(0)をpcに，末尾をスタックポインタとしておく
  - RUNNABLEにする

まとめ

- xv6のプロセス
  - コンピュータの起動
  - メモリレイアウト
  - カーネルの起動
  - スケジューラ
  - 割り込みの仕組み
    - 割り込みハンドラ
    - トランポリンとトラップフレーム
  - プロセスの生成
    - fork / init