日曜研究室

技術的な観点から日常を綴ります

[xv6 #10] Chapter 1 – The first process – Code: entry page table

テキストの19〜20ページ

本文

PCの電源が入ったとき、初期化処理が実行され（BIOSレベルの話）、そしてブートローダがディスクからメモリに読み込まれ、それが実行される。
詳細は付録Bにある。
（付録Bは元のテキストの最後にあります。）
xv6のブートローダは、xv6のカーネルをディスクから読み込み、entryからそれを実行する。
（entryはentry.Sのentry:の部分。以下にその部分のソースを載せておきます。）
（.Sというソースは、gcc向け（かな？）のアセンブラソースファイルです。）

```
01 # Entering xv6 on boot processor, with paging off.
02 .globl entry
03 entry:
04   # Turn on page size extension for 4Mbyte pages
05   movl    %cr4, %eax
06   orl     $(CR4_PSE), %eax
07   movl    %eax, %cr4
08   # Set page directory
09   movl    $(V2P_WO(entrypgdir)), %eax
10   movl    %eax, %cr3
11   # Turn on paging.
12   movl    %cr0, %eax
13   orl     $(CR0_PG|CR0_WP), %eax
14   movl    %eax, %cr0
15
16   # Set up the stack pointer.
17   movl $(stack + KSTACKSIZE), %esp
18
19   # Jump to main(), and switch to executing at
20   # high addresses. The indirect call is needed because
21   # the assembler produces a PC-relative instruction
22   # for a direct jump.
23   mov $main, %eax
24   jmp *%eax
```

x86のページングハードウェアは、カーネルが開始した当初は有効になっていない。
つまりその間は、仮想アドレスはそのまま物理アドレスに対応付けられる。

ブートローダは、xv6のカーネルを物理アドレス0×00100000のメモリに読み込む。
カーネルを0×80100000（カーネルの命令とデータを置く場所として仮想アドレス上で想定してるのと同じ位置）に読み込まない理由は、物理メモリが少ないマシンでそんな高い物理アドレスは実際には使えないかもしれないからである。
カーネルを、0×00000000ではなく0×00100000に置く理由は、0x000a0000〜0×00100000のアドレス範囲は古いI/Oデバイス用だからである。
カーネルの残りの部分を実行可能にするため、entryは0×00000000から始まる物理アドレスを0×80000000から始まる仮想アドレスに対応付けるためにページテーブルをセットアップする。
（0×80000000はKERNBASEとして、memlayout.hに定義されています。）

entry用のページテーブルはmain.cに定義されている。
（main.cのその部分だけ以下に載せておきます。ただmain.c全体でも115行しかありません。）

```
01 // Boot page table used in entry.S and entryother.S.
02 // Page directories (and page tables), must start on a page
   boundary,
03 // hence the "__aligned__" attribute.
04 // Use PTE_PS in page directory entry to enable 4Mbyte pages.
05 __attribute__((__aligned__(PGSIZE)))
06 pde_t entrypgdir[NPDENTRIES] = {
07   // Map VA's [0, 4MB) to PA's [0, 4MB)
08   [0] = (0) | PTE_P | PTE_W | PTE_PS,
09   // Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)
10   [KERNBASE>>PDXSHIFT] = (0) | PTE_P | PTE_W | PTE_PS,
11 };
```

この初期化処理で、1024個のPTEが2組設定され、ゼロと512（KERNBASE>>PDXSHIFT）でインデクスされる。
ほかのPTE群に対してははゼロのままにする。
これで、両方のPTEをスーパーページ（superpage）として使えるようになる。
スーパーページは、仮想アドレス空間における4MBを対応付ける。
エントリー0は、仮想アドレス上の0×00000000〜0×00400000を、物理アドレス上の0×00000000〜0×00400000に対応付ける。
この対応付けは、entryが低いアドレスで実行されている間だけ必要とされる。
やがてこの対応付けは、消去される。
そのページは、PTE_P（利用可能）、PTE_W（書き込み可能）、PTE_PS（スーパーページ）として設定されている。
このあたりのフラグ類と他のページングハードウェア関連の構造体は全てmmu.hに定義されている。

エントリー512は、仮想アドレスKERNBASE〜KERNBASE+0×00400000を物理アドレス0×00000000〜0×00400000に対応付ける。
このエントリはentryが終わったあと、カーネルによって利用される。

これは、ブートローダがカーネルを読み込んだ実際の低い物理アドレスと、後でカーネルがそれ自身の命令とデータを見つけれるようにするための高い仮想アドレスを対応付ける。
この対応付けは、カーネルの命令とデータを4MBに制限する。

entryの話に戻る。
（この段落は、上に載せたアセンブリ言語で書かれたentryのソースの説明です。）
スーパーページを有効にするために、カーネルはまず、CR_PSEフラグ（ページサイズ拡張）をコントロールレジスタ%cr4に設定することによって、それをページングハードウェアに伝える。
次に、entrypgdirの物理アドレスをコントロールレジスタ%cr3に読み込む。
（main.cのentrypgdirの0番目の内容がcr3に読み込まれるということかな。）
ページングハードウェアはentrypgdirの物理アドレスを知らなければならない。
なぜなら、この段階ではまだページングハードウェアは仮想アドレスの変換の仕方を知らないからである。
entrypgdirというシンボルは、高いメモリ位置を指し示していて、V2P_W0マクロ（memlayout.hに定義されている）物理アドレスを算出するために、仮想アドレスからKERNBASEを引く。
ページングハードウェアを有効にするために、xv6はCR0_PGフラグをコントロールレジスタ%cr0に設定する。
CR0_WPもまた設定してるが、それはPTEでカーネルがライトプロテクトフラグを与える事を確実にするためである。

ページング後もまだプロセッサが低いアドレスで命令を実行し続ける事ができる。
それは、entrypgdirが低いアドレスを対応付るから可能なのである。
もしxv6が、entrypgdirからエントリー0を省略したら、コンピュータの世界は、ページングを有効にしたあと命令を実行しようとしたときにクラッシュするだろう。

それからentryは、カーネルのコード（C言語で書かれた分）を転送する必要がある。
そして、それを高いメモリ位置で実行する必要がある。
まず、スタックポインタを作らなければならない。
Cのコードが動作するようにするために、レジスタ%espでスタックを指し示すようにする。
（entryのmovl $(stack + KSTACKSIZE), %espの部分）
低いメモリ位置に対する対応付けが消去されてもスタックが正常に動作するために、全てのシンボルは、スタックを含み高いアドレスを持つ。
最後に、entryは高いアドレスにあるmain関数にジャンプする。
間接的なジャンプが必要とされるのは、アセンブラは直接的なジャンプ（低いメモリバージョンのmain関数を実行する）を生成してしまうからである。
main関数は戻らない。
スタック上で戻る所がないからである。
以上のようにして、カーネルはmain関数の高いアドレスの中で実行される。

感想

今回難しくて訳があやしいところが今までよりたくさんあります。

概要的には、以下の流れかと思います。
1.ブートローダが読み込まれ実行される
2.ブートローダが物理メモリのはじめの方にカーネルを読み込み実行
3.ページテーブル作って、カーネル自体の物理アドレスによる動作を仮想アドレスによる動作へシフト
4.main関数の中のループでOSが動き続ける

で、ややこしいのは3の部分があるからかなと思います。
何せカーネルが自分自身で自分自身の動作基盤となるアドレスを物理から仮想に置き変えるわけです。
しかし逆に言えば、今回ややこしかった部分も3の一文に概念上はまとめれるのかなと思います。

今回出てきたアセンブリ言語で書かれた部分（entry）は、基本的に値を読んでレジスタへセットの繰り返しとそのあとのmain関数へのジャンプのみです。
具体的に何をしてるのかは本文だけでは難しかったのでレジスタ – OS Project Wikiを参照したら分かりやすかったです。

実は本文では、KERNBASE>>PDXSHIFT = 960として書いてありました。
KERNBASEは0×80000000でPDXSHIFTが22ですから、512にならないとおかしいです。
ソースの変更履歴を調べたら、去年の8月末にKERNBASEが0xf0000000から現在の値に変更されてました。
なので960は当時の名残で今は512が正しいはずなのでそれで書いてます。

main.cで定義されているentrypgdirですが、かなり特殊な書き方に見えるけど、要は以下と文の構造は同じです。

```
1 int testarr[10] = {
2     [0] = 123,
3     [5] = 456
4 };
```

これは言うなれば配列の部分初期化で、この例だと0番目が123、5番目が456、ほかは0に初期化されます。
（C言語のバージョンやコンパイラに依存するかも）

本文にあるhigh memoryとかlow addressとかをそのまま高い低いと訳しましたが、なんか全然しっくりきません。
なんかいい言い方ないかな〜。
もしかしたら何か勘違いしてる気もします。

今はなんとなく1日1節を心がけてますが、ちょっとこの難しさだと1節を数日掛けてやるとかになっていくかもしれません。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/2/15 水曜日 [http://peta.okechan.net/blog/archives/1265] |