V0.6改动日志：增加了TLB和虚拟内存部分的一些

因为我还没怎么搞懂phase3要干嘛，这份东西可能有很多地方会写错，希望看出错误的朋友能立刻找我指正，谢谢！

先要修改proj3.conf文件，把内存改大一点，不然没做虚拟内存之前会跑不动

## 关于TLB

为了方便，先把Processor.java里的const贴出来

/\*\* Caused by a syscall instruction. \*/

**public** **static** **final** **int** *exceptionSyscall* = 0;

/\*\* Caused by an access to an invalid virtual page. \*/

**public** **static** **final** **int** *exceptionPageFault* = 1;

/\*\* Caused by an access to a virtual page not mapped by any TLB entry. \*/

**public** **static** **final** **int** *exceptionTLBMiss* = 2;

/\*\* Caused by a write access to a read-only virtual page. \*/

**public** **static** **final** **int** *exceptionReadOnly* = 3;

/\*\* Caused by an access to an invalid physical page. \*/

**public** **static** **final** **int** *exceptionBusError* = 4;

/\*\* Caused by an access to a misaligned virtual address. \*/

**public** **static** **final** **int** *exceptionAddressError* = 5;

/\*\* Caused by an overflow by a signed operation. \*/

**public** **static** **final** **int** *exceptionOverflow* = 6;

/\*\* Caused by an attempt to execute an illegal instruction. \*/

**public** **static** **final** **int** *exceptionIllegalInstruction* = 7;

接下来我们还是要找自信：让程序跑起来

注意网页上说了TLB是fully associative的，如果大家对计算机组成里的cache知识还记得的话，在这个cache里找东西其实是顺序找的，所以。。随便实现一下就行了。。

另：看了下Machine.processor().getTLBSize()，发现输出4，湿了。。。

Track一下就能发现，引发TLBmis的方法是Processor.java里的

**private** **int** translate(**int** vaddr, **int** size, **boolean** writing)

在这里面我们又能track到一个有用的方法

然后就能开始写了，下面这段就已经能跑单进程的程序啦

**int** vpn = *pageFromAddress*(vaddr);

**protected** **void** putTLBEntry(**int** addr) {

//Lib.hdebug(Processor.pageFromAddress(addr) + "!!");

Machine.*processor*().writeTLBEntry((*kk*++) % 4, pageTable[Processor.*pageFromAddress*(addr)]);

}

/\*\*

\* Handle a user exception. Called by <tt>UserKernel.exceptionHandler()</tt>

\* . The <i>cause</i> argument identifies which exception occurred; see the

\* <tt>Processor.exceptionZZZ</tt> constants.

\*

\* **@param** cause

\* the user exception that occurred.

\*/

**public** **void** handleException(**int** cause) {

Processor processor = Machine.*processor*();

**switch** (cause) {

**case** Processor.*exceptionTLBMiss*:

putTLBEntry(Machine.*processor*().readRegister(Processor.*regBadVAddr*));

//Lib.hdebug(Machine.processor().readRegister(Processor.regBadVAddr) + " TLB mis!!" + "pageTableSize : " + pageTable.length);

**break**;

**default**:

**super**.handleException(cause);

**break**;

}

}

大家可能会奇怪我这里为什么写(kk++)，而不直接写到index 0，我本来也以为全部赋给0就行了，但发现不对，原因有点复杂：我track了一下，原因可能如下：在一条指令里会读两次TLB（一条指令会读很多东西吧），所以不能只用一个index。。不然永远会出现TLB mis。。。

这样之后就能跑单进程的程序了，大家可以跑跑看。

为了跑多进程的程序，首先我们要把proj3.conf里的Processor.numPhysPages改的大一些，但之后应该还需要改回来，这里只是为了让程序跑起来。

为了让多线程跑起来，最简单的做法就是每次都刷新TLB，把下面这段东西copy过来就能运行多进程程序了

**public** **void** saveState() {

// super.saveState();

**for** (**int** i = 0;i < Machine.*processor*().getTLBSize();i++)

Machine.*processor*().writeTLBEntry(i, **new** TranslationEntry());

}

/\*\*

\* Restore the state of this process after a context switch. Called by

\* <tt>UThread.restoreState()</tt>.

\*/

**public** **void** restoreState() {

// super.restoreState();

**for** (**int** i = 0;i < Machine.*processor*().getTLBSize();i++)

Machine.*processor*().writeTLBEntry(i, **new** TranslationEntry());

}

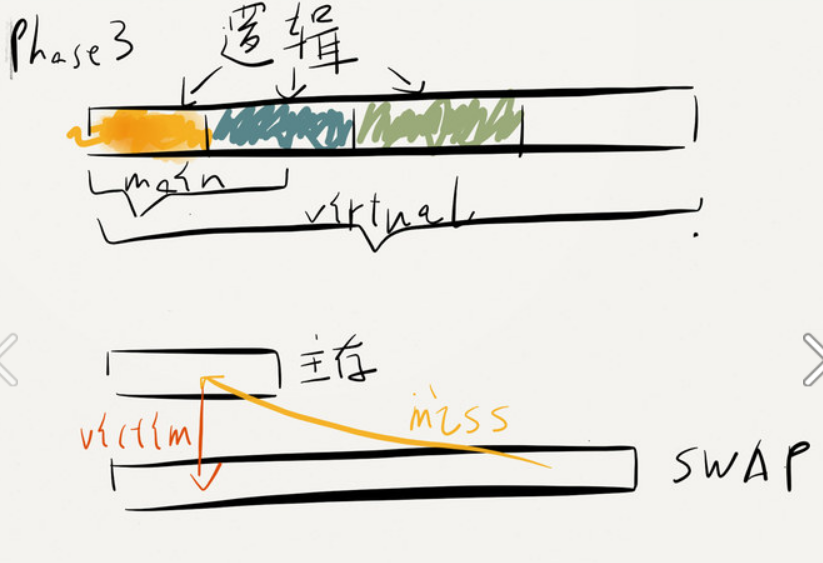
至于，怎么处理TLBmis之类的问题，就要大家自己想了。

注意：

在进程切换时也要注意TLB，这点应该是在VMProcess的saveState和restoreState里做的。

## 关于虚拟内存

到这里我们就要把刚开始扩大的内存给改回来了，因为我们要做虚存了。



大家要注意，这个phase中其实processor完全是在跟TLB打交道，所以我们只要处理好TLB就行了，处理器不会再和PageTable交互（当然我们自己还是可以用pageTable来更新TLB）

另外，网页上也说了，这个phase只需要修改vm包里的两个程序，所以别的就别改了，我们可以把父类的方法给override掉。

这个task的主要结构我觉得是这样的：

一个映射会把逻辑地址映射到虚拟的主存地址（这个是我们在phase2做的事情）

下一个映射会把虚拟的主存地址映射到实际的物理地址（可能是真的主存也可能是硬盘空间）

直接用一个映射把逻辑映到物理似乎也行。。。但我还是喜欢用两个，因为要处理主存与物理之间的交换，似乎用两个会稍微方便点。至于为什么会出现第二个映射，主要是因为我们会swap两个页，第二个映射其实就是告诉进程那个页被换到了哪里

我这段话可能说的很含糊，因为程序注释里已经出现了很多次virtual什么了，那里的virtual我就理解为逻辑地址了，如果你看不懂我说的话就自己写吧，我搞东西有时候会易化神奇。。。

注意1：readTLB那个方法返回的是一个TLB的克隆，所以你想改的话（设置dirty什么的），要用write把那里覆盖掉

注意2：在最后的terminate()里面，把要做的事情放在前面

*swapFile*.close();

ThreadedKernel.*fileSystem*.remove(*swapFileName*);

**super**.terminate();

不然super.terminate()会直接把机器关了。。。

注意3：曾经一直很纠结handleExit里面该不该写Machine.halt，现在不能这样了，因为我们再kernel的terminate里面还有点事情要做，所以现在应该写成

**if** (--*existing* == 0) {

UserKernel.*kernel*.terminate();

}

注意4：在你做SWAP页的时候，请加个锁（各种read，load什么的都要加个锁）。我因为这个调了一天，然后发现居然明明已经disableinterrupt了进程还在乱切。。。。

注意5：

不光我们会给TLB设置dirty，这个方法也会设置dirty

主要方法是Processor.java里面的

private int translate(int vaddr, int size, boolean writing)

这里面会给用到TLB设置dirty

总之不光我们自己会设置dirty

小贴士：

有时候你希望父类能调用子类的方法（因为super的关系this指针变成了父类）

就比如我们想在UserProcess里面调用VMProcess的方法

这时大家可以这样做

**if** (Machine.*processor*().hasTLB()) {

VMProcess kk = (VMProcess)**this**;

kk.acquireVp(pages.get(i));

}