Nachos用户程序和虚拟内存

一、Nachos 对内存、寄存器以及CPU的模拟

Nachos 寄存器组模拟了全部 32 个 MIPS 机（R2/3000）的寄存器，同时加上有关 Nachos 系统调试用的 8 个寄存器，以期让模 拟更加真实化并易于调试，对于一些特殊的寄存器说明如下：

寄存器名	编号	描述
StackReg	29	用户程序的堆栈指针
RetAddrReg	31	存放过程调用的返回地址
HiReg	32	存放乘法结果的高32位
LoReg	33	存放乘法结果的低32位
PCReg	34	当前PC指针
NextPCReg	35	下一条执行语句的PC指针
PrevPCReg	36	上一条执行语句的PC指针（调试用）
LoadReg	37	需要延迟载入的寄存器编号
LoadValueReg	38	需要延迟载入的寄存器值
BadAddReg	39	当出错陷入（Exception）时用户程序的逻辑地址

Nachos 用宿主机的一块内存模拟自己的内存。**每个内存页的大小同磁盘扇 区的大小相同，而整个内存的大小远远小于模拟磁盘的大小。**事实上，Nachos 的内存只有当需要执行用户程序时用户程序的一个暂存地，**而作为操作系统内部使用的数据结构不存放在 Nachos 的模拟内存中**，而是申请 Nachos 所在宿主机的内存。所以 Nachos 的一些重要的数 据结构如线程控制结构等的数目可以是无限的，不受 Nachos 模拟内存大小的限制。

这里需要强调的是，此处 Nachos 模拟的寄存器组同 Thread 类中的 machineState[]数组表示的寄存器组不同，后者代表的是宿主机的寄存器组，是实际存在的； 而前者只是为了运行拥护程序模拟的。

在用户程序运行过程中，会有很多系统陷入核心的情况。系统陷入有两大类原因：**进行系统调用陷入和系统出错陷入。**系统调用陷入在用户程序进行系统调用时发生。系统调用可以看作是软件指令，它们有效地弥补了机器硬件指令不足；系统出错陷入在系统发生错误时发生， 比如用户程序使用了非法指令以及用户程序逻辑地址同实际的物理地址映射出错等情况。不同的出错陷入会有不同的处理，比如用户程序逻辑地址映射出错会引起页面的重新调入，而用户程序使用了非法指令则需要向用户报告等等。Nachos 处理的陷入有：

SyscallException	系统调用陷入
PageFaultException	页面转换出错
ReadOnlyException	试图访问只读页面
BusErrorException	总线错，转换用户程序页面时出错
AddressErrorException	页面访问没有对齐，或者超出了页面的大小
OverflowException	加减法时整数溢出
IllegalInstruException	非法指令访问

模拟机的机器指令由操作代码和操作数组成的，其类定义和实现如下所示：

```
class Instruction {
  public:
    void Decode();              //  将指令的二进制表示转换成系统方便处理的表示
    unsigned int value;         //  指令的二进制表示
    char opCode;                //  分析出的操作代码
    char rs, rt, rd;            //  分析出的指令的三个寄存器的值
    int extra;                  //  分析出的指令立即数
};
```

Machine 类的定义和实现如下所示：

```
class Machine {
  public:
    Machine(bool debug);                          //  初始化方法
    ~Machine();                                   //  析构方法
    void Run();                                   //  运行一个用户程序
    int ReadRegister(int num);                    //  读出寄存器中的内容
    void WriteRegister(int num, int value);       //  向一个寄存器赋值
    void OneInstruction(Instruction *instr);      //  执行一个用户程序指令
    void DelayedLoad(int nextReg, int nextVal);   //  执行一次延迟载入
    bool ReadMem(int addr, int size, int* value); //  读出内存 addr 地址中的内容
    bool WriteMem(int addr, int size, int value); //  向内存 addr 地址中写入内容
    ExceptionType Translate(int virtAddr, int* physAddr, int size,bool writing);
                                                  //  将用户程序逻辑转换成物理地址
    void RaiseException(ExceptionType which, int badVAddr);
                                                  //  执行出错陷入处理程序
    void Debugger();                              //  调用用户程序调试器
    void DumpState();                             //  打印机器寄存器和内存状态
    char *mainMemory;                             //  模拟内存
    int registers[NumTotalRegs];                  //  CPU 寄存器模拟
    TranslationEntry *tlb;                        //  TLB 页面转换表
    TranslationEntry *pageTable;                  //  线性页面转换表
    unsigned int pageTableSize;                   //  线性页面转换表大小
  private:
    bool singleStep;                              //  单步执行标志
    int runUntilTime;                             //  调试时钟
};
```

需要注意的是，虽然这里的很多方法和属性规定为 public 的，但是它们只能在系统核心内被调用。定义 Machine 类的目的是为了执行用户程序，如同许多其它系统一样，用户程序不直 接使用内存的物理地址，而是使用自己的逻辑地址，在用户程序逻辑地址和实际物理地址之 间，就需要一次转换，系统提供了两种转换方法的接口：**线性页面地址转换方法和 TLB 页 面地址转换方法。**

无论是线性页面地址转换还是 TLB 页面地址转换，都需要一张地址转换表，地址转换表是由若干个表项（Entry）组成的。每个表项记录程序逻辑页到内存实页的映射关系，和实页的使用状态、访问权限信息。类 TranslationEntry 描述了表项的结构：

线性页面地址转换是一种较为简单的方式，即用户程序的逻辑地址同实际物理地址之间的关 系是线性的。在作转换时，给出逻辑地址，计算出其所在的逻辑页号和页中偏移量，通过查 询转换表（实际上在使用线性页面地址转换时，**TranslationEntry 结构中的 virtualPage 是多余的**，线性页面转换表的下标就是逻辑页号），即可以得到实际物理页号和其页中偏移量。在模拟机上保存有线性页面转换表，它记录的是当前运行用户程序的页面转换关系；在用户进 程空间中，也需要保存线性页面转换表，保存有自己运行用户程序的页面转换关系。当其被 切换上模拟处理机上运行时，需要将进程的线性页面转换表覆盖模拟处理机的线性页面转换表。

TLB 页面转换则不同，TLB 转换页表是硬件来实现的，表的大小一般较实际的用户程序所 占的页面数要小，所以一般 TLB 表中只存放一部分逻辑页到物理页的转换关系。这样就可 能出现逻辑地址转换失败的现象，会发生 PageFaultException 异常。在该异常处理程序中， 就需要借助用户进程空间的线性页面转换表来计算出物理页，同时 TLB 表中增加一项。如 果 TLB 表已满，就需要对 TLB 表项做 LRU 替换。使用 TLB 页面转换表处理起来逻辑较线 性表为复杂，但是速度要快得多。由于 TLB 转换页表是硬件实现的，所以指向 TLB 转换页 表的指针应该是只读的，所以 Machine 类一旦实例化，TLB 指针值不能改动。

在实际的系统中，线性页面地址转换和 TLB 页面地址转换只能二者取一，目前为简便起见， Nachos 选择了前者，读者可以自行完成 TLB 页面地址转换的实现。

为了解决跨平台的问题，特地给出了四个函数作数据转换， 它们是 WordToHost、ShortToHost、WordToMachine 和 ShortToMachine

1.1 RaiseException 方法

语法： void RaiseException (Exception which, int badVAddr)

参数： which: 系统出错陷入的类型 badVAddr: 系统出错陷入所在的位置

功能： 当系统检查到出错陷入时调用，通过调用 ExceptionHandler 来处理陷入。（见 exception.cc）
ExceptionHandler 目前对所有的陷入都不进行处理，需要进一步加强。 如上面提到的，如果使用 TLB 表时，应该对 PageFaultError 作出处理。

实现： 调用 ExceptionHandler。

1.2 ReadMem 方法

语法： bool ReadMem (int addr, int size, int *value)

参数： addr: 用户程序逻辑地址 size: 需要读出的字节数 value: 读出的内容暂存地

功能： 从用户逻辑地址读出 size 个字节，转换成相应的类型，存放在 value 所指向的空间中。 实现： 调用 Translate 方法。

1.3 WriteMem 方法

语法： bool WriteMem (int addr, int size, int *value)

参数： addr: 用户程序逻辑地址 size: 需要写入的字节数 value: 需要写入的内容

功能： 将 value 表示的数值根据 size 大小转换成相应的机器类型存放在 add 用户逻辑地址中

实现： 调用 Translate 方法。

1.4 Translate方法

语法： Exception Translate (int virtAddr, int *physAddr, int size, bool writing)

参数： virtAddr: 用户程序的逻辑地址 physAddr: 转换后的实际地址 size: 数据类型的大小 writing: 读/写内存标志

功能： 将用户的逻辑地址转换成实际的物理地址，同时需要检查对齐。

实现： 1. 判断用户逻辑地址是否对齐 1.1. 如果 size 是 2，virtAddr 必须是 2 的倍数 1.2. 如果 size 是 4，virtAddr 必须是 4 的倍数
1.3. 没有对齐则返回 AddressErrorException 2. 计算出虚拟地址所在的页号 vpn 及其在页面中的偏移量 3. 根据采用不同的转换方法作不同的处理 3.1. 如果采用的是线性转换表 3.1.1. 当 vpn>=pageTableSize 时，即虚拟页数过大，返回 AddressErrorException 3.1.2. 当页表中显示该页为无效时，返回 PageFaultException 3.1.3. 一切正常则得到相应的页表表项 3.2. 如果采用的是 TLB 转换表，查找 TLB 表 3.2.1. 如果查找到，得到相应的页表表项 3.2.2. 如果没有

查找到，返回 PageFaultException 4. 如果得到的页表表项是只读，writing 标志设置，返回 ReadOnlyException 5. 如果表项中相应的物理地址大于实际的内存物理地址，返回 BusErrorException 6. 设置表项正在使用标志，如果 writing 标志设置，设置表项中的 dirty 标志 7. 返回 NoException