计算机组成与系统结构 实验二

June 9, 2015

简介

本次作业要求基于Y86流水线模拟器实现双核CPU模拟器，并在其上基于共享内存实现双向消息传输程序。

1 L1缓存

本项目在每个处理器上实现了单路L1缓存，每条Cache Line为8个byte，共有16条；每个缓存单元的存储结构包括地址、内容和属性位，定义如下：

```
typedef struct {
  char isValid, isDirty;
  word_t myAddr;
  word_t myContent;
} cache_cell;
```

1.1 缓存策略

每个内存访问请求直接根据地址最低位映射进L1缓存中；如出现冲突，就写回或丢弃旧数据，清空缓存。

1.1.1 Write Allocation

根据项目要求，发生Write Miss时，程序直接在L1缓存中写入新数据，不访问内存。

1.1.2 Write Back

根据项目要求，发生Write Hit时，程序直接在L1缓存中写入数据并标记Dirty，不访问内存。 发生Cache Eviction时，再将数据写入内存。

1.1.3 Read Miss

当某一内存地址出现Read Miss时，L1缓存将同时从内存中读取与之相邻的地址，每次填满整条Cache Line，以利用程序内存访问的Locality提高性能。

2 指令集

为了实现双核程序，我修改了Y86指令集，添加了**I_TESTSET**指令，功能类似于Intel手册中规定的原子指令（**__sync_lock_test_and_set**）。在指令集定义、链接器、模拟器中，皆添加了相应的功能代码。

I_TESTSET指令的流水线分解如下：

```
valP=PC+6
rA:rB=M[PC+1]
valC=M[PC+2]
valB=R[rB]
valE=valB+valC
valM=M[valE]
R[rA]=valM
M[valE]=1
PC=valP
```

在具体实现层面，流水线模拟器在处理此指令时进行了额外的加锁，以保证其原子性。

3 双核架构

使用两个psim进程，各自模拟双核心CPU的一个核心；通过linux系统自带的shm系列函数实现共享整个内存空间；各个CPU仍然有自己各自的寄存器和PC。

在启动时，只有Core0加载了程序二进制文件；Core1由于共享了整个内存空间，也同样可以读取程序代码。

3.1 缓存一致性

为了保证缓存一致性，两个核心之间通过message passing来对内存操作进行同步。

3.1.1 写信息

当出现写指令时，该核心生成一条写信息，包括如下内容：

- 已写地址
- 已写内容

另一核心收到信息后，根据地址查找自己的L1缓存：

- 如缓存中无此条目，忽略此信息。
- 如缓存中有此条目，则更新对应的内容。

3.1.2 读信息

当出现读指令Read Miss时，该核心生成一条读信息，包括如下内容：

- 待读地址

另一核心收到信息后，根据地址查找自己的L1缓存：

- 如缓存中无此条目，忽略此信息。
- 如缓存中有此条目，且是Clean状态，忽略此信息。
- 如缓存中有此条目，且是Dirty状态，则Write Back到主内存。

该核心在确认信息收到后再处理Read Miss，到主内存中读取数据。

3.2 Message Passing技术细节

本项目综合锁、系统信号和共享内存实现了在两个CPU模拟器进程之间进行同步message passing。

3.2.1 共享内存

两个CPU进程共享了如下的内存区域用于寻找对方和发送信息，模拟系统中的消息总线：

```
typedef enum {MSG_READ_WB, MSG_WRITE_SYNC} msg_type;
typedef struct {
  volatile char hasMessage;
  msg_type msgType;
  int msgAddr;
  int msgVal;
  int pid[4];
} system_status;
```

其中，hasMessage用于指示对方需要接收信息，msgType为读或写，msgAddr和msgVal分别为地址和值，pid保存对方的进程编号（用于发送信号）。在消息被接收之后，才能发送下一条消息。

3.2.2 锁

为了避免两个CPU进程同时写入信息，发送信息部分的代码使用Linux系统加锁机制保证互斥性。

3.2.3 信号

在将消息写入共享内存后，进程会等待另一进程确认收到消息（并清除hasMessage变量）；此时，此进程发送SIGUSR1信号给另一进程，另一进程的Signal Handler会立即处理该消息。

由于信号可能丢失，另一进程在写访问L1缓存时，也会检查是否有未处理的消息，并立即处理。

4 乒乓程序

通过汇编编写程序，实现了两个CPU间互相通信，功能简要介绍如下。详细的实现，请参见汇编源代码**pingpong.ys**。

4.1 分程序及分栈

程序启动时，通过对同一内存地址发起Test and Set指令获得不同结果，以区分不同的核心；据此跳转至对应的程序入口，并设置各自的栈指针。具体代码实现如下：

```
init:
  irmovl TESTSET_LOC,%edx
  testset 0(%edx),%eax
  andl %eax,%eax
  je Core0_init
  jmp Core1_init
```

其后，两个核心分别设置不同的栈指针寄存器。

4.2 乒发送

Core0负责Ping方，初始化时从原始程序文件中读取总实验次数，默认为100。

每次发送消息时，通过循环将消息内容（共1到100个数）复制到共享的内存区块MP_Buf中，并求和；之后修改共享的MP_Len变量，告知对方CPU。求和结果写入内存。

在完成消息发送后进入循环，等待MP_Len被清空。

4.3 乓接受

Core1负责Pong方，没有初始数据。

程序初始化后等待MP_Len变量变为非零，之后读取MP_Buf中的数据进行求和，并乘以2作为回复。读取完成后，将MP_Len清空。求和结果也写入内存，可与上一结果比较。

4.4 乒接收

Ping方发现MP_Len被清空后，重新读取MP_Buf中的数据，并进行求和。之后，更改发送消息的长度，重新开始发送阶段。

4.5 结束

当Core0完成所有实验（长度从1到100）后，会清空所有寄存器，并终止程序。

在终止之前，Core0通过共享的内存向Core1发送终止信号；Core1收到后，停止等待新信息，并进行同样的清空和终止操作。