OSTeam19_Report

Team_member：鍾昀諠，陳麒懋

Trace system calls

machine.h

```
enum ExceptionType { NoException,           // Everything ok!
                     SyscallException,      // A program executed a system call.
                     PageFaultException,    // No valid translation found
```

在 ExceptionType 裡有定義一些 instruction 的類別。而這次作業所碰到的則是 SyscallException，當 instruction 指令有用到 Syscall 的話，它會被傳入 RaiseException()裡。

mipssim.cc

首先，先把程式碼透過 Compiler 轉換成 MIPS，然後將它 load 到 machine 來處理。其中在 Machine::Run()裡有 OneInstruction()來處理 MIPS。

```
for (;;) {
    OneInstruction(instr);
    kernel->interrupt->OneTick();
    if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
```

在 OneInstruction()中，假如讀到 SystemCall 指令則會進入 case OP_SYSCALL，然後將 SyscallException 傳入 RaiseException()當中。

```
  case OP_SYSCALL:
RaiseException(SyscallException, 0);
return;
```

machine.cc

之後，產生 interrupts 將 status 由 UserMode 改為 SystemMode。接著將 ExceptionType 傳進 ExceptionHandler()去做處理。做完之後再將 status 變回 UserMode。

```
void
Machine::RaiseException(ExceptionType which, int badVAddr)
{
    DEBUG(dbgMach, "Exception: " << exceptionNames[which]);
    registers[BadVAddrReg] = badVAddr;
    DelayedLoad(0, 0);          // finish anything in progress
    kernel->interrupt->setStatus(SystemMode);
    ExceptionHandler(which);        // interrupts are enabled at this point
    kernel->interrupt->setStatus(UserMode);
}
```

exception.cc

ExceptionHandler()在此檔實作。

Halt()

```
case SC_Halt:
    DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
    SysHalt();
                cout<<"in exception\n";
    ASSERTNOTREACHED();
    break;
```

Create()

```
case SC_Create:
    val = kernel->machine->ReadRegister(4);
    {
    char *filename = &(kernel->machine->mainMemory[val]);
    //cout << filename << endl;
    status = SysCreate(filename);
    kernel->machine->WriteRegister(2, (int) status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
    return;
    ASSERTNOTREACHED();
    break;
```

為了讓 kernel 知道哪個 system call 被呼叫，必須先在 system call 的 interface 也就是 syscall.h 中去定義 SC_Halt,SC_Create 和函數 Halt(), Create()，以方便在 exception.cc 和 start.S 中用到。

- **Start.S**

```
Halt:
    addiu $2,$0,SC_Halt
    syscall
    j   $31
    .end Halt

    .globl MSG
    .ent   MSG
```

```
Create:
    addiu $2,$0,SC_Create
    syscall
    j   $31
    .end Create

    .globl Remove
    .ent    Remove
```

- **ksyscall.h**

ExceptionHandler()之後則是進入 SysHalt()和 SysCreate()。

```
void SysHalt()
{
  kernel->interrupt->Halt();
}
```

```
int SysCreate(char *filename)
{
    // return value
    // 1: success
    // 0: failed
    return kernel->interrupt->CreateFile(filename);
}
```

- **interrupt.h/interrupt.cc**

之後到 interrupt::Halt()和 interrupt::CreateFile()。

```
void
Interrupt::Halt()
{
    cout << "Machine halting!\n\n";
    cout << "This is halt\n";
    kernel->stats->Print();
    delete kernel;  // Never returns.
}
```

```
int
Interrupt::CreateFile(char *filename)
{
    return kernel->CreateFile(filename);
}
```

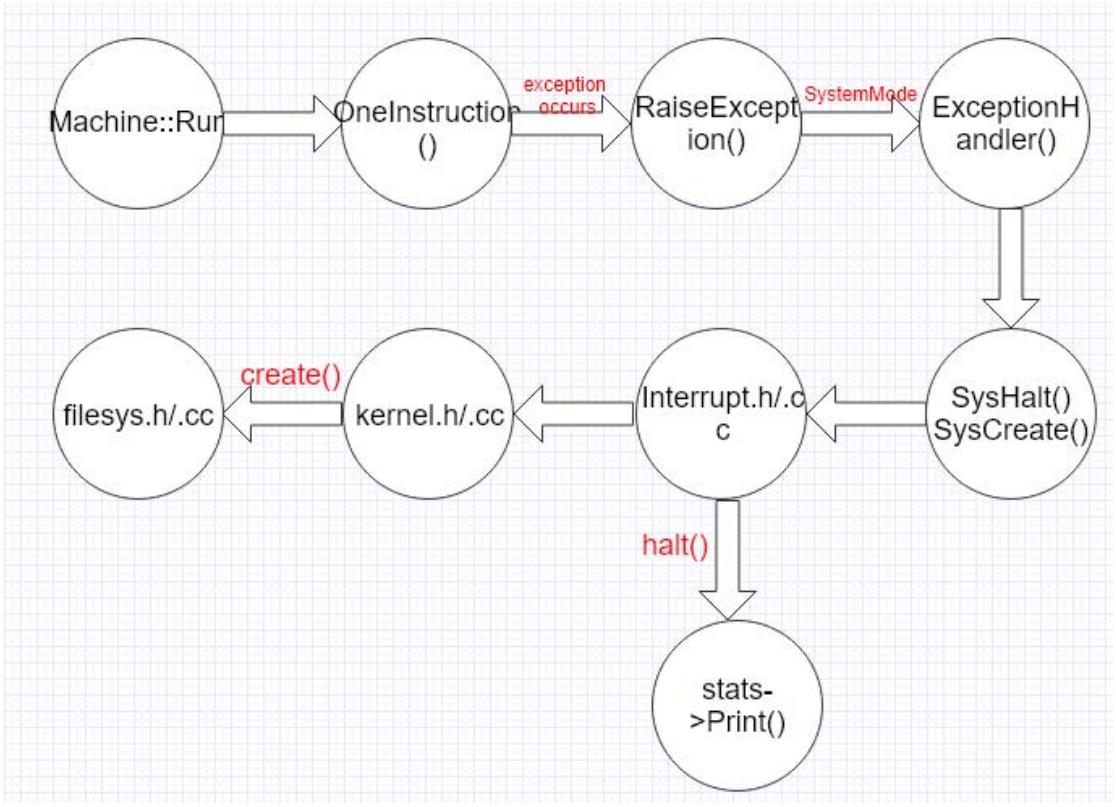
Halt()最後會到 stats.cc 做 Print()的動作。

- **kernel.h/kernel.cc**

進入 Kernel:: CreateFile()然後到 filesys.h/filesys.cc 中實作。

```
int Kernel::CreateFile(char *filename)
{
    return fileSystem->Create(filename);
}
```

Diagram


Machine::Run
OneInstruction()
exception occurs
RaiseException()
SystemMode
ExceptionHandler()
SysHalt()
SysCreate()
Interrupt.h/.cc
kernel.h/.cc
create()
filesys.h/.cc
halt()
stats->Print()