Operating System

Homework1

Cheng-Ying Tsai

1-1. Console I/O system call

```
tests summary: ok:0
9
8
7
6
Machine halting!

Ticks: total 1042, idle 790, system 200, user 52
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 8
Paging: faults 0
Network I/O: packets received 0, sent 0
```

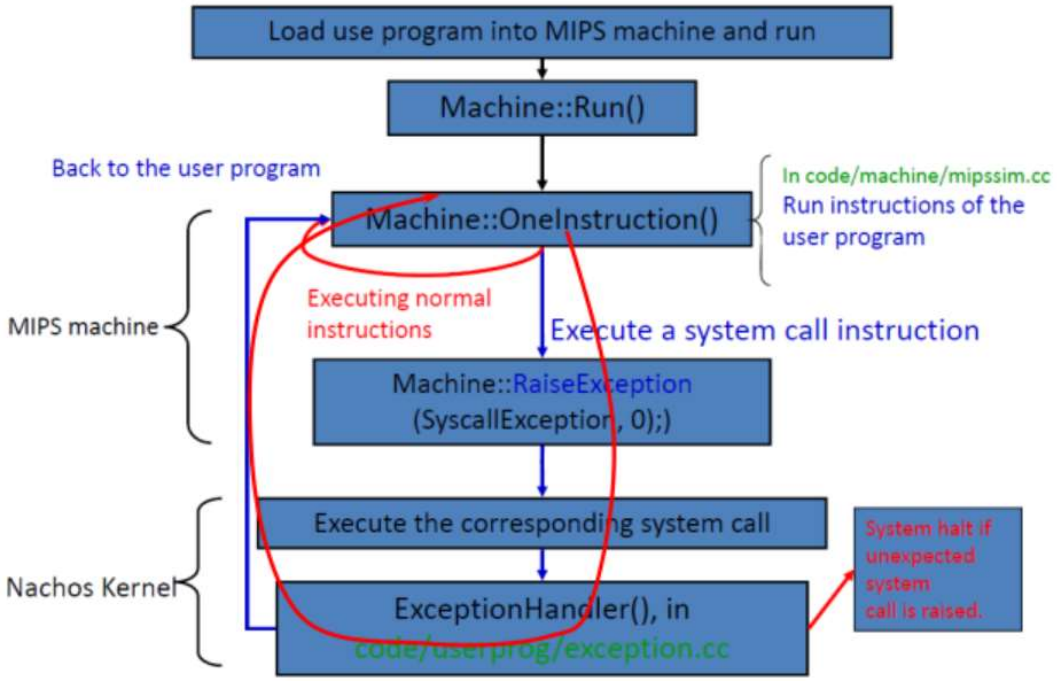
1-2. Implement four file I/O system call

```
chenging@chenging-VirtualBox:~/nachos/NachOS4.1 for ubuntu/code/test$ ../build.l
inux/nachos -x file0_test2.noff


tests summary: ok:0
number = 780
filename = file1.test
number = 1872
buffer =
fileID = 6
fileID = 6
Passed! ^_^
Machine halting!

Ticks: total 169, idle 0, system 30, user 139
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

【以下開始講解 System call 如何發生】



userprog/exeception.cc
ExceptionHandler()

userprog/ksyscall.h
SysPrintInt()

userprog/synchconsole.cc
SynchConsoleOutput::PutInt()
SynchConsoleOutput::PutChar()

machine/console.cc
ConsoleOutput::PutChar()

machine/interrupt.cc
Interrupt::Schedule()

machine/mipssim.cc
Machine::Run()

machine/interrupt.cc
Machine::OneTick()

machine/interrupt.cc
Interrupt::CheckIfDue()

machine/console.cc
ConsoleOutput::CallBack()

userprog/synchconsole.cc
SynchConsoleOutput::CallBack()

1. 當 System Call 發生，首先把函數參數、系統調用存入 Register，然後引發 interrupt 進入 kernel mode。

test/start.s

```
        .globl PrintInt         /*聲明為外部函數*/
        .ent    PrintInt        /*Add函數開始*/

PrintInt:
        addiu $2,$0,SC_PrintInt         /*將system call呼叫case num存入r2暫存器*/
        syscall                         /*所有這個系統的system call會被MIPS machine自動存到暫存器4,5,6,7*/
        j       $31
        .end PrintInt
```

Step I. 將 System call 的 type 存入 2 號暫存器

Step II. 執行 System call 指令

Step III. 返回到 31 號暫存器存放的地址，也就是用戶程式

2. 函數調用則會在打斷時發生，故撰寫發生時的 handler

userprog/exception.cc

```
void
ExceptionHandler(ExceptionType which)
{
    int type = kernel->machine->ReadRegister(2);
        // 從r2取出systemcall type存入type (例如type = SC_PrintInt)
    int number;
    int status, exit, threadID, programID, fileID, numChar;
    DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
    DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which << " type: " << type << ",
<< kernel->stats->totalTicks);

    switch (which) {    // 判斷是system call或其他Exception Type
    case SyscallException:

      switch(type) {    // 判斷system call是什麼type

      case SC_PrintInt:
        {
        DEBUG(dbgSys, "Print Int\n");
        number = kernel->machine->ReadRegister(4);
        DEBUG(dbgTraCode, "In ExceptionHandler(), into SysPrintInt, " << kernel->stats->totalTicks);
        SysPrintInt(number);
        DEBUG(dbgTraCode, "In ExceptionHandler(), return from SysPrintInt, " << kernel->stats-
>totalTicks);
        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;
        }
```

從 4 號暫存器讀入

當 System call “Add”發生時：

Step I. 進入 handler switch 的 case SC_PrintInt

Step II. 呼叫 SysAdd 函數，並將結果存回暫存器 2

Step III. 增加 Program Counter 到下一個準備執行的 instruction

3. 將 system call “SysPrintInt”函數定義

userprog/ksyscall.h

```
void SysPrintInt(int val){
        DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, into synchConsoleOut->PutInt, " << kernel->stats-
>totalTicks);
        kernel->synchConsoleOut->PutInt(val);
        DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, return from synchConsoleOut->PutInt, " << kernel-
>stats->totalTicks);
}
```

userprog/synchConsoleOut.h

```
void PutInt(int number);
```

userprog/synchConsoleOut.cc

```
void SynchConsoleOutput::PutInt(int number){
        char str[15];
    int idx=0;
    sprintf(str, "%d\n\0", number); //simply for trace code

    lock->Acquire();

    do{

        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into consoleOutput->PutChar, " << kernel-
>stats->totalTicks);
    consoleOutput->PutChar(str[idx]);
        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return from consoleOutput->PutChar, " <<
kernel->stats->totalTicks);
        idx++;

        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into waitFor->P(), " << kernel->stats-
>totalTicks);
    waitFor->P();
        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return form  waitFor->P(), " << kernel->stats-
>totalTicks);

    } while (str[idx] != '\0');

    lock->Release();
}
```

Step I. sprintf 將 val 存入 str，變成 char

Step II. lock->Acquire()鎖定物件，執行同步化，只有取得鎖定的 thread 才可以進入同步趨，否則等待，直到取得鎖定

Step III. 將 str 的字元陣列由迴圈方式丟入 consoleOut.cc PutChar()

Step IV. 執行完同步化，lock->Release()解除鎖定

另外 waitFor->P()是讓後面還沒作用的字元先等待

4.

machine/console.cc

```
void
ConsoleOutput::PutChar(char ch)
{
    ASSERT(putBusy == FALSE);
    WriteFile(writeFileNo, &ch, sizeof(char));
    putBusy = TRUE;     // 不能有其它事一起做
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}
```

5. 紀錄 interrupt 何時要被執行，在 Pending Interrupt List 插入要被執行的 interrupt

machine/interrupt.cc

```
void
Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type)
{
    int when = kernel->stats->totalTicks + fromNow;
        // 現在的時間+多久要發生interupt的時間

    PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type);

    DEBUG(dbgInt, "Scheduling interrupt handler the " << intTypeNames[type] << " at time = " << when)
    ASSERT(fromNow > 0);

    pending->Insert(toOccur);
}
```

Argument 分別為，要執行的對象、模擬時間內發生 interrupt 的時間、產生 interrupt 的硬體設備

6. 當系統執行 syscall 指令時，會丟到 mipssim.cc 的 Machine::Run()

machine/ mipssim.cc

```
void
Machine::Run()
{
    Instruction *instr = new Instruction;  // storage for decoded instruction

    if (debug->IsEnabled('m')) {
        cout << "Starting program in thread: " << kernel->currentThread->getName();
        cout << ", at time: " << kernel->stats->totalTicks << "\n";
    }
    kernel->interrupt->setStatus(UserMode);      // syscall happens -> kernel mode
    for (;;) {
        OneInstruction(instr);
        kernel->interrupt->OneTick();  8.
        if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
          Debugger();
    }
}
```

Machine::OneInstruction()模擬 CPU 逐一執行任務功能

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

發現調用了 RaiseException 函數，拋出一個 SyscallException 異常

7.

machine/ machine.cc

```
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);
}
```

ExceptionHandler()結束後才是回到 usermode，搭配 2.再詳讀

machine/ machine.h

```
enum ExceptionType { NoException,           // Everything ok!
                     SyscallException,      // A program executed a system call.
                     PageFaultException,    // No valid translation found
                     ReadOnlyException,     // Write attempted to page marked
                                            // "read-only"
                     BusErrorException,     // Translation resulted in an
                                            // invalid physical address
                     AddressErrorException, // Unaligned reference or one that
                                            // was beyond the end of the
                                            // address space
                     OverflowException,     // Integer overflow in add or sub.
                     IllegalInstrException, // Unimplemented or reserved instr.

                     NumExceptionTypes
};
```

Exception type 的定義如上

8.

machine/ interupt.cc

```
void
Interrupt::OneTick()
{
    MachineStatus oldStatus = status;
    Statistics *stats = kernel->stats;

// advance simulated time
    if (status == SystemMode) {
        stats->totalTicks += SystemTick;
        stats->systemTicks += SystemTick;
    } else {
        stats->totalTicks += UserTick;
        stats->userTicks += UserTick;
    }
    DEBUG(dbgInt, "== Tick " << stats->totalTicks << " ==");

// check any pending interrupts are now ready to fire
    ChangeLevel(IntOn, IntOff); // first, turn off interrupts
                                // (interrupt handlers run with
                                // interrupts disabled)
    CheckIfDue(FALSE);          // check for pending interrupts
    ChangeLevel(IntOff, IntOn); // re-enable interrupts
    if (yieldOnReturn) {        // if the timer device handler asked
                                // for a context switch, ok to do it now
        yieldOnReturn = FALSE;
        status = SystemMode;            // yield is a kernel routine
        kernel->currentThread->Yield();
        status = oldStatus;
    }
}
```

另外第二題的實作比較容易，單純把原先就寫好的 code 接上即可，故這裡就不贅述。