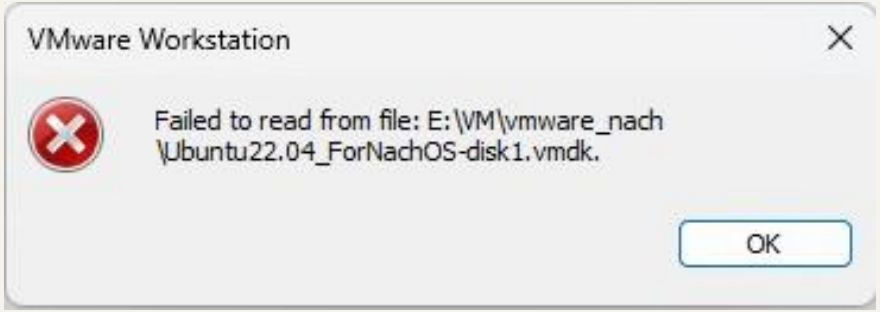
Debug


VMware Workstation
Failed to read from file: E:\VM\vmware_nach
\Ubuntu22.04_ForNachOS-disk1.vmdk.
OK


Ubuntu22.04_ForNachOS-disk1.vmdk
我
2024年10月4日 我
本機 › SSD_750 (E:) › VM › vmware_nach
搜尋 vmware_nach
名稱
修改日期
類型
大小
Ubuntu22.04_ForNachOS.mf 2024/10/7 上午 12:58 MF 檔案 1 KB
Ubuntu22.04_ForNachOS.ovf 2024/10/7 上午 12:58 Open Virtualizati... 15 KB
Ubuntu22.04_ForNachOS-disk1-001.v... 2024/10/7 上午 12:59 VMware virtual d... 2,663,343...
Ubuntu22.04_ForNachOS-file1.iso 2024/10/7 上午 12:59 光碟映像檔 2,055,086...

NachOS - System Call

Assignment

Assignment

- 在本次 Lab 中，必須建立四個 System Call：**Open()、Write()、Read()、Close()**

並且通過 fileIO_test1.c、fileIO_test2.c 的測試

- fileIO_test1.c 會測試 Open()、Write()、Close() 三個 System Call
- fileIO_test2.c 會測試 Open()、Read()、Close() 三個 System Call

Assignment

```
#include "syscall.h"

int main(void) {
    char test[] = "abcdefghijklmnopqrstuvwxyz";
    int success = Create("file1.test");
    OpenFileId fid;
    int i;

    if (success != 1)
        MSG("Failed on creating file");
    fid = Open("file1.test");

    if (fid < 0)
        MSG("Failed on opening file");

    for (i = 0; i < 26; ++i) {
        int count = Write(test + i, 1, fid);
        if (count != 1)
            MSG("Failed on writing file");
    }

    success = Close(fid);
    if (success != 1)
        MSG("Failed on closing file");
    MSG("Success on creating file1.test");
    Halt();
}
```

Assignment

```
#include "syscall.h"

int main(void) {
    // you should run fileIO_test1 first before running this one
    char test[26];
    char check[] = "abcdefghijklmnopqrstuvwxyz";
    OpenFileId fid;
    int count, success, i;
    fid = Open("file1.test");
    if (fid < 0)
        MSG("Failed on opening file");
    count = Read(test, 26, fid);
    if (count != 26)
        MSG("Failed on reading file");
    success = Close(fid);
    if (success != 1)
        MSG("Failed on closing file");
    for (i = 0; i < 26; ++i) {
        if (test[i] != check[i])
            MSG("Failed: reading wrong result");
    }
    MSG("Passed! ^_^");
    Halt();
}
```

Assignment

- 通過測試之畫面如下

```
02:45:18 root@c981b814ad1e test → ../build.linux/nachos -e fileIO_test1
fileIO_test1
Success on creating file1.test
Machine halting!

This is halt
Ticks: total 954, idle 0, system 130, user 824
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
02:45:21 root@c981b814ad1e test → ../build.linux/nachos -e fileIO_test2
fileIO_test2
Passed! ^_^
Machine halting!

This is halt
Ticks: total 815, idle 0, system 120, user 695
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
02:45:22 root@c981b814ad1e test →
```

Assignment

- 實作過程必須遵守以下規定

I. 禁止使用 standard libraries 的任何 IO function (e.g. printf(), cout, fopen(), fwrite(), write(), etc.)。

反之，NachOS 已經準備好對應 Function。

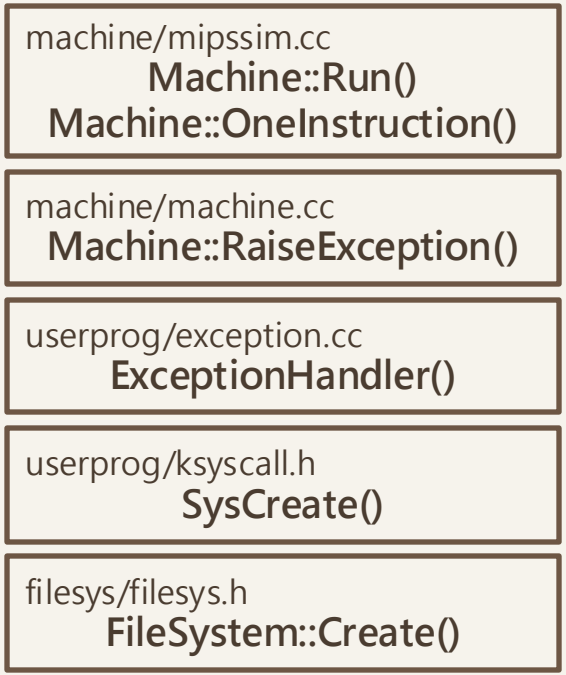
II. 禁止修改 "machine/" 資料夾下任何檔案。

III. 禁止修改 "filesys/" 資料夾下任何檔案。

Hint - Trace Code

Trace Code

- 呼叫 Create() 這個 System Call 系統時，完整運作流程爲何？



Trace Code

- machine/mipssim.cc - **Machine::Run()**

```
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);
    for (;;) {
        DEBUG(dbgTraCode, "In Machine::Run(), into OneInstruction "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        OneInstruction(instr);
        DEBUG(dbgTraCode, "In Machine::Run(), return from OneInstruction  "
                          << "== Tick " << kernel->stats->totalTicks << " ==");

        DEBUG(dbgTraCode, "In Machine::Run(), into OneTick "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        kernel->interrupt->OneTick();
        DEBUG(dbgTraCode, "In Machine::Run(), return from OneTick "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
            Debugger();
    }
}
```

Trace Code

- machine/mipssim.cc - **Machine::Run()**
  - 將系統狀態切換到 User Mode
  - 在無限迴圈中重複執行以下動作：呼叫 OneInstruction()、將時間前進一單位

Trace Code

- machine/mipssim.cc - **Machine::OneInstruction(Instruction *instr)**

```
void Machine::OneInstruction(Instruction *instr) {
#ifdef SIM_FIX
    int byte;  // described in Kane for LWL,LWR,...
#endif

    int raw;
    int nextLoadReg = 0;
    int nextLoadValue = 0;  // record delayed load operation, to apply
                            // in the future

    // Fetch instruction
    if (!ReadMem(registers[PCReg], 4, &raw))
        return;  // exception occurred
    instr->value = raw;
    instr->Decode();
```

```
case OP_SYSCALL:
    DEBUG(dbgTraCode, "In Machine::OneInstruction, RaiseException(SyscallException, 0), " << kernel->stats->totalTicks);
    RaiseException(SyscallException, 0);
    return;
```

Trace Code

- machine/mipssim.cc - **Machine::OneInstruction(Instruction *instr)**

  - 從記憶體讀取指令（讀到的是組合語言）

    In Machine::Run(), into OneInstruction == Tick 41 ==

    Reading VA 4, size 4

    Translate 4 , read

    phys addr = 4

    value read = 0

    At PC = 4 SLL r0,r0,0

  - 解譯指令

Trace Code

- machine/machine.cc - **Machine::RaiseException(ExceptionType which, int badVAddr)**

```
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);
}
```

- which：何種類型的 Exception（以本作業為例就是 SyscallException）
- badVAddr：位址
- 切換到 System Mode，開始處理 Exception。

Trace Code

- **userprog/exception.cc** - **ExceptionHandler(ExceptionType which)**

```
//----------------------------------------------------------------------
// ExceptionHandler
//      Entry point into the Nachos kernel.  Called when a user program
//      is executing, and either does a syscall, or generates an addressing
//      or arithmetic exception.
//
//      For system calls, the following is the calling convention:
//
//      system call code -- r2
//                arg1 -- r4
//                arg2 -- r5
//                arg3 -- r6
//                arg4 -- r7
//
```

- 根據 NachOS 官方描述，暫存器存放的內容如上。

Trace Code

- **userprog/exception.cc** - **ExceptionHandler(ExceptionType which)**

```
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;
```

- 每一個 Exception 都是一個 case。

如果要新增一個 System Call，第一個步驟為何？

Trace Code

- **userprog/exception.cc** - **ExceptionHandler(ExceptionType which)**

```
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;
```

- Create 的使用方式為 bool Create(char *name) ，因此要從 r4 讀入第一個參數。

  如果有多個參數？

Trace Code

- **userprog/exception.cc** - **ExceptionHandler(ExceptionType which)**

```
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;
```

- 從 Main Memory 取得 filename，並且調用函式創建檔案。
- 將狀態寫回暫存器 r2。

Trace Code

- **userprog/exception.cc** - **ExceptionHandler(ExceptionType which)**

```
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;
```

- 將當前的 Program Counter 寫入至 PrevPCReg（Previous Program Counter）。
- 將當前的 Program Counter + 4 寫入至 PCReg（移動到下一條指令）。
- 將再下一條指令的位置寫入至 NextPCReg。

Trace Code

- **userprog/ksyscall.h** - SysCreate(char *filename)

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

Trace Code

- **filesys/filesys.h** - Create(char *name)

```
bool Create(char *name) {
    int fileDescriptor = OpenForWrite(name);

    if (fileDescriptor == -1)
        return FALSE;
    Close(fileDescriptor);
    return TRUE;
}
```

Hint - Implement system call

Implement system call

- 根據前一章節所述，實作 System Call 時需要修改的檔案如下
  - userprog/exception.cc
  - userprog/ksyscall.h

- 除此之外，另有二個檔案需要進行修改
  - test/start.s
  - userprog/syscall.h

Implement system call

- **test/start.s**

```
/* ------------------------------------------------------------
 * System call stubs:
 *      Assembly language assist to make system calls to the Nachos kernel.
 *      There is one stub per system call, that places the code for the
 *      system call into register r2, and leaves the arguments to the
 *      system call alone (in other words, arg1 is in r4, arg2 is
 *      in r5, arg3 is in r6, arg4 is in r7)
 *
 *      The return value is in r2. This follows the standard C calling
 *      convention on the MIPS.
 * ------------------------------------------------------------
 */
```

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

Assembler Directives：以 '.' 為開頭，用於告訴 Assembler 執行操作、宣告資料轉換型態。

Implement system call

- **test/start.s**

```
/* ------------------------------------------------------------------
 * System call stubs:
 *      Assembly language assist to make system calls to the Nachos kernel.
 *      There is one stub per system call, that places the code for the
 *      system call into register r2, and leaves the arguments to the
 *      system call alone (in other words, arg1 is in r4, arg2 is
 *      in r5, arg3 is in r6, arg4 is in r7)
 *
 *      The return value is in r2. This follows the standard C calling
 *      convention on the MIPS.
 * ------------------------------------------------------------------
 */
```

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

.globl <symbol>：讓 Linker 可以看到 <symbol>，其他 program 也能使用這個指令。

.ent <symbol>：同 .type，可以將 <symbol> 標記為函數、數據、或定義為其他屬性。

.end <symbol>：結束 Assembly file。

Implement system call

- **userprog/syscall.h**

```
#include "copyright.h"
#include "errno.h"
/* system call codes -- used by the stubs to tell the kernel which system call
 * is being asked for
 */
#define SC_Halt 0
#define SC_Exit 1
#define SC_Exec 2
#define SC_Join 3
#define SC_Create 4
#define SC_Remove 5
```

Implement system call

- test/Makefile

```
INCDIR =-I../userprog -I../lib
CFLAGS = -g -G 0 -c $(INCDIR) -B/usr/bin/local/nachos/lib/gcc-lib/decstation-ultrix/2.95.2/ -B/
usr/bin/local/nachos/decstation-ultrix/bin/

ifeq ($(hosttype),unknown)
PROGRAMS = unknownhost
else
# change this if you create a new test program!
PROGRAMS = add halt createFile LotOfAdd fileIO_test1 fileIO_test2
endif
```

Implement system call

- 實作此 Lab 時之建議：

  I. 可以根據 Trace Code 章節推斷出如何實作 System call。

  II. 每個檔案都會有註解說明，請詳閱註解。

  III. 如果想要自行 Trace Code，可以善用 NachOS 的 Debug mode。

  ../build.linux/nachos -d + -e [YourProgram]

  ../build.linux/nachos -d + -e createFile

  IV. 網路資源很多，但同學機測時**必須能詳細說明撰寫的程式碼意義為何**否則不予計分。

END