NachOS　HW1: System call

106030028 劉多聞
107030028 劉騏鋒
1060300　 陳致瑋

Part 0 Introduction to NachOS

1. What is NachOS?

2. What is the system architecture of NachOS

Part I. Trace code.

Halt()

1. Machine::Run()
   (1) 切換成 user mode
   (2) 然後執行無限 for loop (逐條執行 userprogram 的程式指令)
   (3) 進 OneInstruction()
2. Machine::OneInstruction()
   OneInstruction 針對每一條 userprogram 指令的 opCode 有對應的 case 處理動作
   (1) 進 OP_SYSCALL 的 case (會產生 halt 勢必因為有呼叫 system call 所以 opcode 的 case 是 OP_SYSCALL)
   (2) 進 RaiseException(SyscallException, 0)
3. Machine::RaiseException()
   (1) 將 0 填進 BadVAddrReg 暫存器 (The name stands for Bad Virtual Address, which contain the memory address where the exception has occurred.) 在此與 Bad Virtual Address 無關，故值為零
   (2) 接著將 kernel 切換為 system mode
   (3) 下一步進 ExceptionHandler() 結束後切換為 user mode
4. ExceptionHandler()
   (1) 根據 RaiseException 傳過來的 ExecutionType 做對應的處置，而取到的為 SyscallException。
   (2) 知道 ExecutionType 為 SyscallException 後再到對應的暫存器 (對應的暫存器為 2，2 是讀取 system call 種類的地方) 取值。根據 userprog/syscall.h，Halt 值為 0，於是進入”SC_Halt” case。
   (3) 進 SC_Halt 後執行 interrupt 裡的 Halt 程式。
5. Interrupt::Halt()

(1) 輸出”Machine halting”，並印出來
(2) 關閉 kernel

Create():

1. Machine::Run()
   (1) 切換成 user mode
   (2) 然後執行無限 for loop (逐條執行 userprogram 的程式指令)
   (3) 進 OneInstruction()
2. Machine::OneInstruction()
   OneInstruction 針對每一條 userprogram 指令的 opCode 有對應的 case 處理動作
   (1) 進 OP_SYSCALL 的 case (會產生 halt 勢必因為有呼叫 system call 所以 opcode 的 case 是 OP_SYSCALL)
   (2)進 RaiseException(SyscallException, 0)
3. Machine::RaiseException()
   (1) 將 0 填進 BadVAddrReg 暫存器 (The name stands for Bad Virtual Address, which contain the memory address where the exception has occurred.) 在此與 Bad Virtual Address 無關，故值為零
   (2) 接著將 kernel 切換為 system mode
   (3) 下一步進 ExceptionHandler() 結束後切換為 user mode
4. ExceptionHandler()
   (1) 根據 RaiseException 傳過來的 ExecutionType 做對應的處置，而取到的為 SyscallException。
   (2) 知道 ExecutionType 為 SyscallException 後再到對應的暫存器 (對應的暫存器為 2，2 是讀取 system call 種類的地方) 取值。根 userprog/syscall.h，Create 值為 4，於是進入”SC_Create” case。
   (3) 進 SC_Create 後執行 machine.cc 裡的 Machine::ReadRegister(int num=4)程式。
   (4) 在 Machine::ReadRegister(int num=4)中會用 ASSERT 這個函式檢查(num>=0 && num < NumTotalRegs)，之後回傳讀取暫存器的值(register[num=4])存入 val 變數中。
   (5) 進入 mainMemory 讀取 filename 位址
   (6) 根據 filename 指標進入 filesys.h 呼叫 Create()
   (7) 做完 Create()函式後，將得到的 status 丟進 WriteRegister()中，WriteRegister 做的事情是將值寫入 user program register 裡面。
5. FileSystem::Create()
   (1) 呼叫 C++函式 OpenForWrite()確認是否有檔案開啟準備來寫入值，若有則值為 true，若無則為 false

(2) Close()函式來關閉檔案

6. Machine::OneInstruction()

(1) Program Counter 的實作方式是用 Register[]這個矩陣中儲存當前應執行的指令

(2) 每當執行完一次指令會透過不同的變數計算 Program Counter 接下來要執行哪個指令

Part II. Please implement four system calls of file system operation below.

A. Code Implemtation

1. exception.cc

```
//<TODO
case SC_Open:
        val = kernel->machine->ReadRegister(4);
        {
                char *filename = &(kernel->machine->mainMemory[val]);
                status = kernel->fileSystem->OpenAFile(filename);
                kernel->machine->WriteRegister(2, (int)status);
        }
        return;
        ASSERTNOTREACHED();
        break;
case SC_Write:
        val = kernel->machine->ReadRegister(4);
        {
                char *buffer = &(kernel->machine->mainMemory[val]);
                status = kernel->fileSystem->WriteFile(buffer, kernel->machine->ReadRegister(5));
                kernel->machine->WriteRegister(2, (int)status);
        }
        return;
        ASSERTNOTREACHED();
        break;
case SC_Read:
        val = kernel->machine->ReadRegister(4);
        {
                char *buffer = &(kernel->machine->mainMemory[val]);
                status = kernel->fileSystem->ReadFile(buffer, kernel->machine->ReadRegister(5));
                kernel->machine->WriteRegister(2, (int)status);
        }
        return;
        ASSERTNOTREACHED();
        break;

case SC_Close:
        val = kernel->machine->ReadRegister(4);
        {
                status = kernel->fileSystem->CloseFile();
                kernel->machine->WriteRegister(2, (int)status);
        }
        return;
        ASSERTNOTREACHED();
        break;
//TODO>
```

2. start.s

```
//<TODO
        .globl Open
        .ent    Open
Open:
        addiu $2,$0,SC_Open
        syscall
        j       $31
        .end Open

        .globl Read
        .ent    Read
Read:
        addiu $2,$0,SC_Read
        syscall
        j       $31
        .end Read

        .globl Write
        .ent    Write
Write:
        addiu $2,$0,SC_Write
        syscall
        j       $31
        .end Write

        .globl Close
        .ent    Close
Close:
        addiu $2,$0,SC_Close
        syscall
        j       $31
        .end Close

//TODO>
```

3. syscall.h

```
//<TODO
#define SC_Open         5
#define SC_Write        6
#define SC_Read         7
#define SC_Close        8
//TODO>
```

```
//<TODO
int OpenAFile(char *name);
/* Open the Nachos file "name", and return an "OpenFileId" that can
 * be used to read and write to the file.
 */
int WriteFile(char *buffer, int size);
/* Write "size" bytes from "buffer" to the open file. */
int ReadFile(char *buffer, int size);
/* Read "size" bytes from the open file in;to "buffer".
 * Return the number of bytes actually read -- if the open file isn't
 * long enough, or if it is an I/O device, and there aren't enough
 * characters to read, return whatever is available (for I/O devices,
 * you should always wait until you can return at least one character).
 */
int CloseFile();
/* Close the file, we're done reading and writing to it. */

//TODO>
```

4. filesys.h

```
//<TODO
//The OpenAFile function is used for kernel open system call

OpenFile* filePtr;      //you need to use this filePtr to manage the current file

int OpenAFile(char *name){
        filePtr = Open(name);
        if(filePtr == NULL)return -1;
        else return 1;
}
int WriteFile(char *buffer, int size){
        if(filePtr == NULL)return -1;
        return filePtr->Write(buffer, size);
}
int ReadFile(char *buffer, int size){
        if(filePtr == NULL)return -1;
        return filePtr->Read(buffer, size);
}
int CloseFile(){
        if(filePtr == NULL)return -1;
        else{
                Close(filePtr->getFile());
                return 1;
        }
}

//TODO>
```

B. Compile and Test

1. Compile:

   make

   make clean

```
dorianliu@dorianliu-VirtualBox: ~/nachos-master-nachos-4.0-hw1/nachos-4.0-hw1/code
../bin/coff2noff test2.coff test2
numsections 3
Loading 3 sections:
        ".text", filepos 0xd0, mempos 0x0, size 0x160
fice Calc ".data", filepos 0x230, mempos 0x160, size 0x0
        ".bss", filepos 0x0, mempos 0x160, size 0x0
/usr/local/nachos/decstation-ultrix/bin/gcc -G 0 -c -I../userprog -I../threads -I../lib -I../userprog -I../threads -I..
/lib  -c -o fileTest1.o fileTest1.c
/usr/local/nachos/decstation-ultrix/bin/ld -T script -N start.o fileTest1.o -o fileTest1.coff
../bin/coff2noff fileTest1.coff fileTest1
numsections 4
Loading 4 sections:
        ".text", filepos 0xf0, mempos 0x0, size 0x1b0
        ".rdata", filepos 0x2a0, mempos 0x1b0, size 0x60
        ".data", filepos 0x300, mempos 0x210, size 0x0
        ".bss", filepos 0x0, mempos 0x210, size 0x0
/usr/local/nachos/decstation-ultrix/bin/gcc -G 0 -c -I../userprog -I../threads -I../lib -I../userprog -I../threads -I..
/lib  -c -o fileTest2.o fileTest2.c
/usr/local/nachos/decstation-ultrix/bin/ld -T script -N start.o fileTest2.o -o fileTest2.coff
../bin/coff2noff fileTest2.coff fileTest2
numsections 4
Loading 4 sections:
        ".text", filepos 0xf0, mempos 0x0, size 0x310
        ".rdata", filepos 0x400, mempos 0x310, size 0xb0
        ".data", filepos 0x4b0, mempos 0x3c0, size 0x0
        ".bss", filepos 0x0, mempos 0x3c0, size 0x0
make[1]: Leaving directory `/home/dorianliu/nachos-master-nachos-4.0-hw1/nachos-4.0-hw1/code/test'
dorianliu@dorianliu-VirtualBox:~/nachos-master-nachos-4.0-hw1/nachos-4.0-hw1/code$ userprog/nachos -e test/fileTest1
Total threads number is 1
```

2. test :

   1. userprog/nachos -e test/fileTest1

```
dorianliu@dorianliu-VirtualBox:~/nachos-master-nachos-4.0-hw1/nachos-4.0-hw1/code$ userprog/nachos -e test/fileTest1
Total threads number is 1
Thread test/fileTest1 is executing.
Success :)
Machine halting!

Ticks: total 78, idle 0, system 30, user 48
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
```

2. userprog/nachos -e test/fileTest2