OS Homework1 Report

Team21

葉宥忻 109062301 - Open, Write

陳禹勳 109062134 - Read, Close

Trace code

(a)

當我們initialize kernel後，會執行kernel->ExecAll()，產生一個thread來run我們的program，再經過開啟user program、初始化register、page table等操作後，接下來開始執行Machine::Run()

Machine::Run()

當user program開始執行的時候，準備好一個Instruction的data structure，並把status設成user mode，在for迴圈裡 OneInstruction一個一個讀並執行指令，每個指令執行完後呼叫OneTick()增加simulated time，同時檢查有沒有interrupt需要處理

Machine::OneInstruction()

從memory中fetch instruction進行decode，再根據opcode判斷他是甚麼指令做相應的操作(如add、jal等)，並且更新pc，如果有error或是syscall則會把exceptionType(例如SyscallException)當作參數傳給RaiseException()

Machine::RaiseException()

紀錄造成trap的virtual address，並從user mode轉換到kernel mode，讓os得以處理exception，並把ExceptionType繼續傳給ExceptionHandler()，處理完再跳回user mode

ExceptionHandler()

ExceptionHandler()接收到exceptionType後，從register2判斷是哪種syscall type，做相應的處理，如遇到SC_halt就會做SysHalt()，最後更新pc

SysHalt()

定義了os如何處理SC_Halt，也就是呼叫kernel->interrupt->Halt()

Interrupt::Halt()

印出一些資訊(如Ticks、PageFults number等等)，並終止掉整個NachOS

(b)

ExceptionHandler()

接收到exceptionType後，做相應的處理，如遇到SC_Create就做SysCreate(filename)，而filename會放在mainMemory[val]中，val在register[4]中，最後更新pc

SysCreate()

定義了os怎麼去處理SC_Create，也就是呼叫kernel->fileSystem->Create()，若成功create回傳1，否則回傳0

FileSystem::Create()

1. 確認directory有沒有重複名字的file
2. 確認disk有沒有空間給file header
3. 確認directory有沒有空間給file name跟file header
4. 確認disk有沒有空間給data
5. 若1~4都成功，則把header、更新過的freemap、更新過的directory寫回disk，並return True；失敗則return False。

(c)

ExceptionHandler()

接收到exceptionType後，做相應的處理，如遇到SC_PrintInt就做SysPrintInt(val)，val放在register[4]裡面，最後更新pc

SysPrintInt()

定義了os怎麼去處理SC_PrintInt，也就是呼叫kernel->synchConsonleOut->PutInt(val)

SynchConsoleOutput::PutInt()

1. 把int轉成str

2. lock -> Acquire(): 避免和其他thread同時output
3. consoleOutput -> PutChar(): 把char寫進display
4. waitFor->P(): 等待後面呼叫了CallBack()中的waitFor->V()表示output結束
5. 迴圈重複執行3、4，一個一個output str的char
6. lock->Release(): thread釋出lock

SynchConsoleOutput::PutChar()

1. lock -> Acquire(): 避免和其他thread同時output
2. consoleOutput -> PutChar(): 把char寫進display
3. waitFor->P(): 等待後面呼叫了CallBack()中的waitFor->V()表示output結束
4. lock->Release(): thread釋出lock

ConsoleOutput::PutChar()

透過putBusy限制一次write一個char，使用WriteFile函式來write到display上，因為ConsoleOutput的writeFile==NULL，所以writeFileNo=1，因此我們會output在console上，最後再schedule ConsoleWriteInt這個interrupt在經過ConsoleTime時間後。

Interrupt::Schedule()

將interrupt依據什麼時間要執行，新增一個PendingInterrupt到pending這個list中。

Machine::Run()

當user program開始執行的時候，準備好一個Instruction的data structure，並把status設成user mode，在for迴圈裡 OneInstruction一個一個讀並執行指令，每個指令執行完後呼叫OneTick()增加simulated time，同時檢查有沒有interrupt需要處理

Machine::OneTick()

看現在是user mode還是kernel mode增加simulated time，先disable interrupt，用CheckIfDue()去檢查並處理interrupt，再enable interrupt回來，最後檢查是否要context switch。

Interrupt::CheckIfDue()

1. 先確認有沒有pending interrupt
2. 先看pending中的第一個interrupt，如果已經到了執行的時間，或是

advanceClock=True直接advance時間，去執行他的CallBack()，接下來依序取出其他時間到了的interrupt，去呼叫他們的CallBack()。

ConsoleOutput::CallBack()

putBusy設回Flase，更新kernel->stats->numConsoleCharsWritten，呼叫callWhenDone->CallBack()，也就是SynchConsoleOutput::CallBack()。

SynchConsoleOutput::CallBack()

呼叫waitFor->V()，告知完成output，可以進行下一個char的output了。

Implement four I/O system calls

Start.S:

在Start.S中新增Open、Write、Read、Close的assembly code

Open:

在exceptionHandler中新增syscall type為SC_Open的case，把name的pointer放在mainMemory[val]中，其中val為放在register[4]的值。

在ksyscall.h新增OpenFileId SysOpen()，OpenFileId指的是OpenFile在OpenFileTable上的ID，若成功Open則回傳放在Table上的ID，失敗則回傳-1。而SysOpen()會往下呼叫OpenAFile()，OpenAFile的實作方法是：先檢查有沒有此name的file若沒有則回傳-1，若有再用for迴圈看OpenFileTable上有沒有空位(NULL)，有空位則把此空位給此OpenFile，沒空位則回傳-1。

Write:

在exceptionHandler中新增syscall type為SC_Write的case，從memory和register中取得buffer、size和OpenFileId，呼叫SysWrite，在ksyscall.h新增int SysWrite(char *buffer, int size, OpenFileId id) function return kernel->fileSystem->WriteFile(buffer, size, id)，WriteFile function呼叫OpenFileTable[id]->Write，也就是OpenFile::Write，OpenFile::Write呼叫了OpenFile::WriteAt，讓我們從檔案的起始位置寫入指定的字數，最後回傳真實寫入的字數，如果失敗回傳-1，例如OpenFileId invalid

Read:

在exceptionHandler中新增syscall type為SC_Read的case，從memory和register中取得buffer、size和OpenFileId，呼叫SysRead，在ksyscall.h新增int SysRead(char *buffer, int size, OpenFileId id) function return kernel->fileSystem->ReadFile(buffer, size, id)，ReadFile function呼叫OpenFileTable[id]->Read，也就是OpenFile::Read，OpenFile::Read呼叫了OpenFile::ReadAt，讓我們從檔案的起始位置讀取指定的字數，最後回傳真實讀到的字數，如果失敗回傳-1，例如OpenFileId invalid

Close:

在exceptionHandler中新增syscall type為SC_Close的case，從register中取得OpenFileId，呼叫SysClose，在ksyscall.h新增int SysClose(OpenFileId id) return kernel->fileSystem->CloseFile(id)，在CloseFile function中Delete OpenFileTable[id]，OpenFile的destructor就會呼叫Close(file)去關閉檔案，最後再將OpenFileTable[id]設為NULL，成功就回傳1，失敗回傳-1，例如OpenFileId invalid

Difficulties

1. 一開始對nachos的架構還有lock的方法不太懂，有些地方不知道為什麼會這樣跑，但是trace更多code後，就比較清楚了
2. SynchConsoleOutput跟ConsoleOuput的CallBack分不太清楚，Trace完他們的Constructor後，就比較知道他們的關係
3. 在做OpenAFile的時候，不小心把 if(OpenFileTable[i] == NULL)打成 if(OpenFileTable[i] = NULL，導致一直無法把OpenFile成功放進 OpenFileTable。後來藉由更改其return的值來觀察他是從哪邊return出來的，發現是錯在條件的判斷