OS_MP1_Report

組員：趙仰生、郭蕙綺
貢獻：50%、50%

part(b)
SC_Halt：

Machine::run()：
這個function讓kernel去開始usermode，為了模擬user-level的計算（離開kernel-level），在無限迴圈裡，執行一個user的instruction（呼叫OneInstruction()），並且送出onetick的interrupt。

Machine::OneInstruction()：
這個function用來執行由run()呼叫的instruction，而當interrupt或是exception發生時，呼叫machine裡的RaiseException()，並且，當此function return時，會回到run()，並且重新一個instruction，這也是為什麼在run()裡設計成無限迴圈，當state發生改變時，可以重新執行instruction，且必須計算下一個instruction的位置(pc += 4)，讓下一個instruction可以馬上執行。

Machine::RaiseException()：
這個function用於，當user呼叫system call(interrupt)或是發生exception時，將控制權轉移到kernel，並且呼叫ExceptionHandler()。

ExceptionHandler()：
這個function先確認exception kind是SyscallException後，根據其種類來執行相對應的事情：
SC_Halt：呼叫SysHalt()。
SC_PrintInt：呼叫SysPrintInt(val) 其中value要到machine裡的函數取值後set program counter再return。
SC_MSG：呼叫SysHalt()。
SC_Create：呼叫SysCreate(filename)去更改status後set program counter再return。
SC_Add：呼叫SysAdd(int1, int2)去更改result後set program counter再return。
SC_Exit：呼叫Finish()來終止目前的thread。

SysHalt()：
這個function在ksyscall.h標頭檔裡面（kernel介面，用來處理systemcalls），從kernel呼叫Halt()。

Interrupt::Halt()：
這個function用來終止Nachos，也會印出性能的統計。

SC_Create：

ExceptionHandler()：
這個function先確認exception kind是SyscallException後，根據其種類來執行相對應的事情：
SC_Halt：呼叫SysHalt()。
SC_PrintInt：呼叫SysPrintInt(val) 其中value要到machine裡的函數取值後set program counter再return。
SC_MSG：呼叫SysHalt()。
SC_Create：呼叫SysCreate(filename)去更改status後set program counter再return。
SC_Add：呼叫SysAdd(int1, int2)去更改result後set program counter再return。
SC_Exit：呼叫Finish()來終止目前的thread。

SysCreate()：
這個function在ksyscall.h標頭檔裡面（kernel介面，用來處理systemcalls），從kernel呼叫FileSystem::Create()，並且return 1 或 0 的值（為了告知ExceptionHandler有沒有成功Create，記錄在status裡）。

FileSystem::Create()：
這個function呼叫OpenForWrite()，並且回傳 True 或 False 給SysCreate（SysCreate會再回傳給ExceptionHandle，如同上述）。

SC_PrintInt：

ExceptionHandler()：
這個function先確認exception kind是SyscallException後，根據其種類來執行相對應的事情：
SC_Halt：呼叫SysHalt()。
SC_PrintInt：呼叫SysPrintInt(val) 其中value要到machine裡的函數取值後set program counter再return。
SC_MSG：呼叫SysHalt()。
SC_Create：呼叫SysCreate(filename)去更改status後set program counter再return。
SC_Add：呼叫SysAdd(int1, int2)去更改result後set program counter再return。
SC_Exit：呼叫Finish()來終止目前的thread。

SysPrintInt() :
這個function在ksyscall.h標頭檔裡面（kernel介面，用來處理systemcalls），從kernel呼叫SynchConsoleOutput::PutInt(val)。

SynchConsoleOutput::PutInt()：
這個function先用lock鎖住，再將數字轉為字串後呼叫consoleOutput->PutChar()將字串顯示在顯示器上，並且呼叫waitFor在必要時等待，最後再將lock釋放。

SynchConsoleOutput::PutChar()：
這個function先用lock鎖住，呼叫consoleOutput->PutChar()將字串顯示在顯示器上，並且呼叫waitFor在必要時等待，最後再將lock釋放。

ConsoleOutput::PutChar() :
這個function將字串寫入file後，在指定時間排程interrupt後return。

Interrupt::Schedule() :
這個function主要目的是要在指定時間（現在+等待時間）插入interrupt。值得注意的是，kernel不應該直接呼叫他，而是要讓hardware device呼叫。

Machine::run()：
這個function讓kernel去開始usermode，為了模擬user-level的計算（離開kernel-level），在無限迴圈裡，執行一個user的instruction（呼叫OneInstruction()），並且送出onetick的interrupt。

Machine::OneTick()：
這個function用來提前模擬時間並檢查是否有任何待處理的interrupt被called，而導致OneTick被called的情況有兩種：重新啟用interrupt，執行user的instruction。
檢查pending interrupt的流程：
先turn off interrupt，然後檢查pending interrupt，再重新啟動interrupt。

Interrupt::CheckIfDue() :
這個function的主要目的是要確認有沒有interrupt正在等待發生，當觸發interrupt handler時回傳true，而當沒有pending interrupt或未來有interrupt（pending的第一個觸發時間>totalTicks時）但ready queue還有其他東西時，回傳false。
將stats->idleTicks加上pending裡第一個interrupt的時間扣掉totalTicks的時間，而新的totalTicks會等於pending裡第一個interrupt的時間。
最後將第一個interrupt拉出list且呼叫interrupt handler後return true。

ConsoleOutput::CallBack()：
當下一個字符可以輸出到顯示器時，這個function將被模擬器呼叫，並且將numConsoleCharsWritten+1。

SynchConsoleOutput::CallBack()：
這個function用於，當可以安全地發送下一個字符時，呼叫interrupt handler，確保可以發送到顯示器。

part(c)
start.S:
當我們call system call時，通常要包一個給C/C++使用的wrapper，才可以從C/C++裡呼叫syscall，這個檔案主要就是在做這件事情，實作caller端本身，宣告完global跟ent後，總共有三個步驟，第一個就是將$2 = $0 + 該system call的編號(在syscall.h檔定義)，第二就是進到system call handler處理該system call，最後jump return 到 return register，也就是return。（我們只要將題目要求的四種system call加上去就行了）
syscall.h:
這個標頭檔包含兩部分，第一部分是我們要修改的地方，也就是定義system call種類的編號(例如#define SC_Open 6)，這樣kernel才會知道要怎麼處理這個system call，第二部分是已經寫好的地方，也就是system call的介面，主要讓user可以在C/C++上面呼叫system call。
exception.cc、ksyscall.h、filesys.h（依照system call來解釋）：
Open：
在test1.c檔中呼叫Open函數時傳file的檔名，從exceptionHandler實作SC_Open，先用ReadRegister取arg1的值（字串記憶體位址），再到mainMemory要filename後呼叫ksyscall的Sysopen並將字串傳入，Sysopen再呼叫filesys的OpenAFile，在OpenAFile裡先利用for-loop找尋fileDescriptorTable裡NULL的位置（id），如果超過20還沒有NULL表示table已滿，或者是OpenForReadWrite時出錯就return -1，成功的話則在NULL那格 new 一個OpenFile 後return id，層層傳回ExceptionHandler後寫回system call的位置，最後調整program counter的位置（PC+=4）。

Write：
在test1.c裡，是利用for-loop呼叫Write，並傳入char陣列的位址、size（每次寫1個）及呼叫完open回傳的fid值，在ExceptionHandler實作SC_Write，利用ReadRegister取arg1的值（字串記憶體位址），再到mainMemory要字串後呼叫ksyscall的Syswrite，將字串、size(arg2)、fid(arg3)值傳入，Syswrite再呼叫filesys的WriteFile，進來後要先檢查能不能寫入（fileDescriptorTable[id]等於NULL時、寫入size小於0時、id超出table的範圍時無法寫入），不能寫入return -1，可以的話則呼叫openfile裡的Write，回傳的值為numbytes，層層回傳ExceptionHandler後寫回system call的位置，最後調整program counter的位置（PC+=4）。

Read：
在test1.c裡，是直接呼叫Read，並傳入char陣列的位址、size及呼叫完open回傳的fid值，在ExceptionHandler實作SC_Read，利用ReadRegister取arg1的值（字串記憶體位址），再到mainMemory要字串後呼叫ksyscall的Sysread，將字串、size(arg2)、fid(arg3)值傳入，Sysread再呼叫filesys的ReadFile，進來後要先檢查能不能讀取（fileDescriptorTable[id]等於NULL時、寫入size小於0時、id超出table的範圍時無法讀取），不能讀取return -1，可以的話則呼叫openfile裡的Read，回傳的值為numbytes，層層回傳ExceptionHandler後寫回system call的位置，最後調整program counter的位置（PC+=4）。

Close：

在test1.c裡，呼叫close是直接傳入fid值，因此在ExceptionHandler實作SC_Close時直接呼叫ksyscall的Sysclose，將fid值傳入，Sysclose再呼叫filesys的CloseFile，進來後要先檢查能不能關閉（fileDescriptorTable[id]等於NULL時、id超出table的範圍時無法關閉），不能關閉return -1，可以的話則直接delete fileDescriptorTable[id]並將其設為NULL後return 1（表示成功），層層回傳ExceptionHandler後寫回system call的位置，最後調整program counter的位置（PC+=4）。

part(d)、part(e)

首先是在trace code完之後，有些細節其實還沒有了解的很深入，所以在實際要打code的時候，會不知道該如何實際運用函數，不過在查了些資料，並且閱讀其他檔案（例如openfile.h、sysdep.cc）之後，就了解如何操作了。

比較值得提到的是，在寫exception.cc的時候，在不同case裡面如果要宣告相同變數，需要用大括號包起來。

這次作業花了蠻久的時間，主要是在一開始trace code跟閱讀其他檔案的部分，但是因為做完了這次的作業，我們更深入了解到OS在實際執行上的架構與操作，例如：memory該如何讀取、該如何呼叫interrupt、該如何運用table來實際執行system call等等，也了解到上學期為什麼要學計算機結構了。