● Requirements:
Include the following answers in your writing report:
(a). Explain the purposes and details of each function call listed in the code path above.
(b). Explain how the arguments of system calls are passed from user program to kernel in each of the above use cases.

模板:
(a)
(1)purpose:
(2)detail:

(b)

SC_Halt

Machine::Run()
(a)
(1)purpose:模擬硬體在Nachos上執行binary code。
(2)detail:執行已編譯好的program，並且在執行一條instruction之後，interrupt一次，讓OS可以管理硬體的使用分配。
(b)
No arguments from the user program.

Machine::OneInstruction(`Instruction *instr`**)**
(a)
(1)purpose:模擬CPU執行binary code。
(2)detail:先呼叫ReadMem函數判斷是否有exception出現，若沒有exception，將創建一個instruction instance，將binary code decode成instruction，並判斷各種opCode的情況，根據不同opcode做相應的運算。

(b)
No arguments from the user program. (instr是在Machine::Run()創建的)

Machine::RaiseException (`ExceptionType which, int badVAddr`**)**

(a)
(1)purpose: 把user mode切成kernel mode，處理exception(如除以0，或者記憶體的error或system call)
(2)detail: 當OneInstruction讀到有exception時，就會呼叫RaiseException()，清掉load register的index與value，並將kernel設為system mode，呼叫ExceptionHandler()執行該exception，做完後切換為user mode。

(b) user_program連結到syscall.h，還有Start.S把syscall.h的function連結到kernel對應的function，接著compiler把user program編譯成binary code，最後從binary code解讀出"which"和"BadVAddr"。

ExceptionHandler(ExceptionType which)
(a)
(1)purpose:在kernel mode執行exception instruction。
(2)detail: 從registers[2](被定義在Start.S)讀取exception type，根據不同type取得不同的arguments，並做不同的exception處理(不過目前只有處理這個作業用到的system call)。
補充(a)(2):
SC_Halt: 呼叫SysHalt()
SC_PrintInt: 讀出registers[4](argument1)，並呼叫SysPrintInt(registers[4])
SC_MSG: 讀出registers[4](argument1)，從main memory取出訊息並印出，接著呼叫SysHalt()，終止程式運行。
SC_Create: 讀出registers[4](argument1)，並從main memory讀取檔案名稱，呼叫SysCreate()並寫入檔案開啟狀態到registers[2]，最後更新PCreg。
SC_Add: 呼叫SysAdd，Sys_Add會把registers[4](argument1)、registers[5](argument2)加起來存進registers[2](return value)，最後更新PCreg。

(b)
(user_program連結到syscall.h，還有Start.S把syscall.h的function連結到kernel對應的function，接著compiler把user program編譯成binary code，最後從binary code解讀出"which"和"BadVAddr"，接著which傳入ExceptionHandler)。

SysHalt()
(a)
(1)purpose: 將Exception Handler的SC_Halt case連接到interrupt::Halt()
(2)detail: 呼叫kernel->interrupt->halt，進入Halt()。

(b) No arguments.

Interrupt::Halt()
(a)
(1)purpose: 關掉kernel。
(2)detail: 印出關掉kernel的訊息及kernel當前狀態，最後delete kernel。

(b) No arguments.

SC_Create

ExceptionHandler(ExceptionType which)
(a)
(1)purpose:在kernel mode執行exception instruction。
(2)detail: 從registers[2](被定義在Start.S)讀取exception type，根據不同type取得不同的arguments，並做不同的exception處理(不過目前只有處理這個作業用到的system call)。
補充(a)(2):
SC_Halt: 呼叫SysHalt()
SC_PrintInt: 讀出registers[4](argument1)，並呼叫SysPrintInt(registers[4])
SC_MSG: 讀出registers[4](argument1)，從main memory取出訊息並印出，接著呼

叫SysHalt()，終止程式運行。
SC_Create: 讀出registers[4](argument1)(檔案名稱在memory的地址)，並從main memory讀取檔案名稱，呼叫SysCreate()並寫入檔案開啟狀態到registers[2]，最後更新PCreg。
SC_Add: 呼叫SysAdd，Sys_Add會把registers[4](argument1)、registers[5](argument2)加起來存進registers[2](return value)，最後更新PCreg。

(b)
user_program連結到syscall.h，還有Start.S把syscall.h的function連結到kernel對應的function，接著compiler把user program編譯成binary code，最後從binary code解讀出"which"和"BadVAddr"，接著which傳入ExceptionHandler()。

SysCreate(`char *filename`)

(a)
(1)purpose: 將Exception Handler的SC_Create case連接到FileSystem::Create()
(2)detail: 呼叫kernel->filesystem的Create()，創建後，return create 成功與否(0:失敗,1:成功)

(b)
在Exception Handler執行時，register[4]會被抓到(char*) filename裡面，接著傳入這個function。而register[4]的由來是，compiler編譯過user和kernel code之後，把user code轉成binary code，再由OneInstruction轉譯成instruction，在運行`Create("file0.test");`之後，`"file0.test"`會做為function的argument存在registers[4]。

FileSystem::Create(`char *name`)
(a)
(1)purpose: 創建一個檔案
(2)detail: 呼叫sysdep.h的openforwrite()，回傳值大於0代表file的id，並關閉檔案;回傳值等於-1代表創建失敗，回傳false。

(b) 根據上述**SysCreate(`char *filename`)**參數傳輸流程，再從該函數把filename放進**FileSystem::Create(`char *name`)**。

SC_PrintInt
ExceptionHandler()
(a)
(1)purpose:在kernel mode執行exception instruction。
(2)detail: 從registers[2](被定義在Start.S)讀取exception type，根據不同type取得不同的arguments，並做不同的exception處理(不過目前只有處理這個作業用到的system call)。
補充(a)(2):
SC_Halt: 呼叫SysHalt()
SC_PrintInt: 讀出registers[4](要print的整數值)，並呼叫SysPrintInt(registers[4])
SC_MSG: 讀出registers[4](argument1)，從main memory取出訊息並印出，接著呼叫SysHalt()，終止程式運行。

SC_Create: 讀出registers[4](argument1)，並從main memory讀取檔案名稱，呼叫SysCreate()並寫入檔案開啟狀態到registers[2]，最後更新PCreg。
SC_Add: 呼叫SysAdd，Sys_Add會把registers[4](argument1)、registers[5](argument2)加起來存進registers[2](return value)，最後更新PCreg。

SysPrintInt(`int val`)
(a)
(1)purpose:連接**Exception Handler**與kernel的**SynchConsoleOutput::PutInt()**

(2)detail:呼叫kernel的SynchConsoleOutput::PutInt()
(b)
在Exception Handler執行時，register[4]會被抓到val裡面，接著傳入這個function。而register[4]的由來是，compiler編譯過user和kernel code之後，把user code轉成binary code，再由OneInstruction轉譯成instruction，在`result = Add(42, 23);`之後，result會做為function的argument存在registers[4]。

SynchConsoleOutput::PutInt(`int value`)
(a)
(1)purpose: 把整數轉換成字串，讓**SynchConsoleOutput::PutChar(`char ch`)**輸出到console。
(2)detail:先拿到Semaphore(互斥鎖)的控制權，把value的值寫入str的字元指標，持續呼叫consoleOutput->PutChar()和idx++直到str[idx] 等於 '\0'。並把鎖的控制權釋放。
(b)根據上述的**SysPrintInt(`int val`)**參數傳輸流程，將val從該函數傳入本函數。

SynchConsoleOutput::PutChar(`char ch`)
(a)
(1)purpose:輸出一個字元到console，如有需要則等待。
(2)detail:先拿到Semaphore(互斥鎖)的控制權，呼叫consoleOutput->PutChar(ch)，呼叫P()(等待互斥鎖值 > 0，檢查互斥鎖的值(檢查時不能做content switch，所以我們先disable interrupt)，釋放互斥鎖控制權。
(b) **command line的argv->`kernel->ConsoleTest()->synchConsoleIn->GetChar()`->本函數。**

ConsoleOutput::PutChar(`char ch`)
(a)
(1)purpose:輸出並替kernel安排未來interrupt
(2)detail: 輸出一個字元到模擬display，安排一個interrupt，並return。
(b)有兩種情況:
1.從**SynchConsoleOutput::PutInt(`int value`)**傳入，將value轉換成字串，字串的每個元素會一一傳入本函數。
2. 根據上述，接著從**SynchConsoleOutput::PutChar(`char ch`)**傳到本函數。

Interrupt::Schedule(`CallBackObj *toCall, int fromNow, IntType type`**)**
(a)
(1)purpose:輸出並替kernel安排未來interrupt
(2)detail: 輸出一個字元到模擬display，安排一個interrupt，並return。
(b)
callback function是一個需要排程的interrupt function，**fromNow**是紀錄排程當時的時間點，**type**是interrupt的function type。三者皆非user program傳入。

Machine::Run()
(a)
(1)purpose:模擬硬體在Nachos上執行binary code。
(2)detail:先把模式切換成user mode，之後不斷呼叫One Instruction和interrupt的One Tick函數。
(b) No arguments from the user program.

Machine::OneTick()
(a)
(1)purpose:模擬時間並檢查是否有interrupt事件需要被執行。
(2)detail:先將時間同步化，如果是System mode，則將totalTicks和systemTicks同時加上SystemTick，若非，則將userTicks和systemTicks同時加上UserTick。接著，呼叫ChangeLevel暫停interrupt，呼叫CheckIfDue，檢查是否有任何等待中的interrupt，接著恢復執行interrupt。如果timer想要context switch，則執行context switch，並更改狀態。
(b) No arguments from the user program.

Interrupt::CheckIfDue(`bool advanceClock`**)**
(a)
(1)purpose:如果任何interrupt已經過期，則將之移除。
(2)detail:如果沒有等待中的interrupt，則return False。如果下個事件還沒準備要發生，return False，若準備要發生，則直接將clock設成下個interrupt發生的時間，如果kernel->machine是有意義的。flush掉存register的值，和即將要存在register的值。當pending list 有物件且pending list的首項已經過期，則先清掉首項，再呼叫首項的callback function。
(b) No arguments from the user program.

ConsoleOutput::CallBack()
(a)
(1)purpose:simulator將下一個character 輸出到display。
(2)detail:將output busy狀態改成false，將numConsoleCharsWritten+1，
callWhenDone callback object呼叫Call back function。
(b) No arguments from the user program.

SynchConsoleOutput::CallBack()
(a)
(1)purpose:增加value的值，呼叫在等待的事件。
(2)detail:轉換interrupt的狀態，執行queue首項之事件，value+1，執行re-enable的interrupt。

(b) No arguments from the user program.

3. Report
● Working items
(a). Cover page, including team member list, team member contributions
(b). Explain how system calls work in NachOS as requested in Part II-1.
(c). Explain your implementation as requested in Part II-2.
Compiler會把Part II-2的四個function(user program)、ksyscall.h及Start.S編譯起來，ksyscall.h在compile time會把user program的function引導到kernel，而kernel利用OpenAFile()、ReadFile()、WriteFile()、CloseFile()對應處理他們，compile成instruction後，machine在跑instruction的時候，就會raise exception，接著連接到exception handler，再從register[2]抓下exception type，而下一行instruction就會是kernel對應到的function，以下將分別介紹四個function。

OpenAFile()
利用OpenForReadWrite(檔名)開啟檔案，檔名不存在回傳-1，存在則放入OpenFileTable的一個空位，若Table已滿，則回傳-1。

WriteFile()
尋找OpenFileTable[file_id]，若為NULL(invalid file_id)，回傳-1，若有則呼叫Openfile object的write function，write function會依照給定的字串位置及數量寫入檔案，回傳寫入檔案的數量。不過如果position>file_length，或者size小於0，則會回傳0。

ReadFile()
與Write file相似，讀取OpenFileTable[file_id]，若為NULL(invalid file_id)，回傳-1。若有則呼叫openfile object的read function，此function會回傳讀取到的字元數。

CloseFile()
OpenFileTable[file_id]如果是NULL，代表發生錯誤，回傳-1；如果有東西，那就把OpenFileTable[file_id]清掉，並且回傳1，代表成功。

(d). What difficulties did you encounter when implementing this assignment?
(e). Any feedback you would like to let us know.

(d). What difficulties did you encounter when implementing this assignment?
We had issues understanding fileDescriptor at first, But after reading Tutorials, we

got some ideas. We firstly went to syscall.h and exception.cc file to uncomment and add some cases. Then we dealt with the interface in ksyscall.h. Then we have no difficulties afterwards since we use the correct way and check our code while coding it. And we lose the part of start.S at first. We finally finished it after adding the script at start.S.

(e)Any feedback you would like to let us know?

Pop quiz能不能考投影片上有的，有點太難，因為我很認真上課並且在考前都有複習，但考的題目不在投影片上。