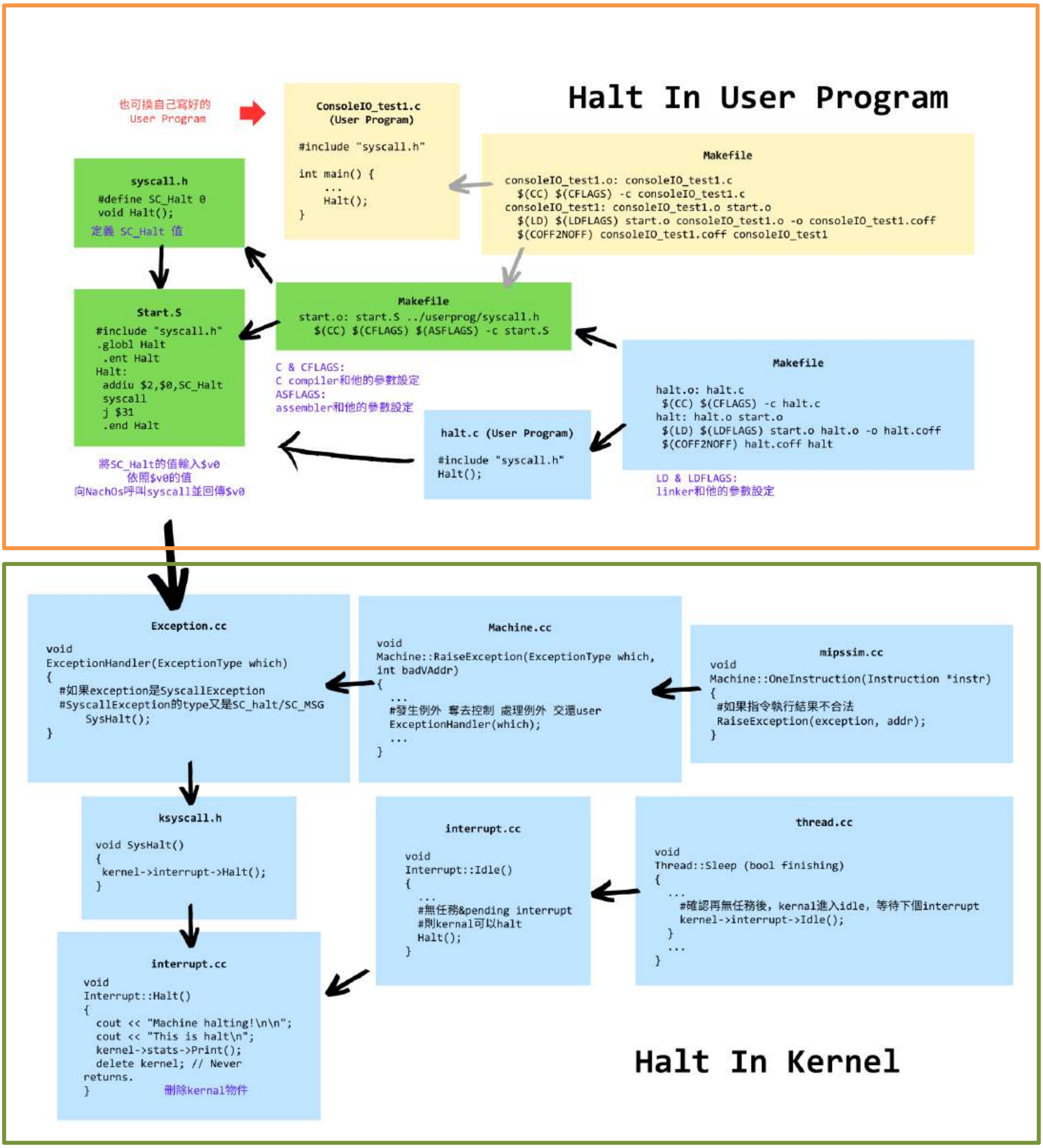
110705017 何翊華 110705063 廖偉辰

Part 1:Trace Code Result

1. Flow Chart of Halt() System Call:



Halt In User Program

也可換自己寫好的 User Program

```
ConsoleIO_test1.c
(User Program)

#include "syscall.h"

int main() {
    ...
    Halt();
}
```

```
Makefile

consoleIO_test1.o: consoleIO_test1.c
  $(CC) $(CFLAGS) -c consoleIO_test1.c
consoleIO_test1: consoleIO_test1.o start.o
  $(LD) $(LDFLAGS) start.o consoleIO_test1.o -o consoleIO_test1.coff
  $(COFF2NOFF) consoleIO_test1.coff consoleIO_test1
```

```
syscall.h
#define SC_Halt 0
void Halt();
```

定義 SC_Halt 值

```
Makefile
start.o: start.S ../userprog/syscall.h
  $(CC) $(CFLAGS) $(ASFLAGS) -c start.S
```

C & CFLAGS:
C compiler和他的參數設定
ASFLAGS:
assembler和他的參數設定

```
Start.S
#include "syscall.h"
.globl Halt
 .ent Halt
Halt:
 addiu $2,$0,SC_Halt
 syscall
 j $31
 .end Halt
```

將SC_Halt的值輸入$v0
依照$v0的值
向NachOs呼叫syscall並回傳$v0

```
Makefile

halt.o: halt.c
 $(CC) $(CFLAGS) -c halt.c
halt: halt.o start.o
 $(LD) $(LDFLAGS) start.o halt.o -o halt.coff
 $(COFF2NOFF) halt.coff halt
```

LD & LDFLAGS:
linker和他的參數設定

```
halt.c (User Program)

#include "syscall.h"
Halt();
```

```
Exception.cc

void
ExceptionHandler(ExceptionType which)
{
  #如果exception是SyscallException
  #SyscallException的type又是SC_halt/SC_MSG
      SysHalt();
}
```

```
Machine.cc
void
Machine::RaiseException(ExceptionType which,
int badVAddr)
{
  ...
  #發生例外 奪去控制 處理例外 交還user
  ExceptionHandler(which);
  ...
}
```

```
mipssim.cc
void
Machine::OneInstruction(Instruction *instr)
{
 #如果指令執行結果不合法
 RaiseException(exception, addr);
}
```

```
ksyscall.h

void SysHalt()
{
 kernel->interrupt->Halt();
}
```

```
interrupt.cc

void
Interrupt::Idle()
{
  ...
  #無任務&pending interrupt
  #則kernal可以halt
  Halt();
}
```

```
thread.cc

void
Thread::Sleep (bool finishing)
{
  ...
    #確認再無任務後, kernal進入idle, 等待下個interrupt
    kernel->interrupt->Idle();
  }
  ...
}
```

```
interrupt.cc
void
Interrupt::Halt()
{
  cout << "Machine halting!\n\n";
  cout << "This is halt\n";
  kernel->stats->Print();
  delete kernel; // Never
returns.
}
```

刪除kernal物件

Halt In Kernel

2. Details of Trace Halt() Code

<u>test/start.S</u>

```
        .globl Halt
        .ent  Halt
Halt:
    addiu $2,$0,SC_Halt    //將 SC_Halt 值(設為 0)存入 reg2 裡
    syscall   //呼叫 system call  觸發  Machine Run()
    j     $31
    .end Halt
```

machine/machine.h

```
// Routines callable by the Nachos kernel
  void Run();           //  觸發  Mipssim Run()
```

machine/Mipssim.cc

```
void Machine::Run()
{
  …
  kernel->interrupt->setStatus(UserMode); //  目前為 UserMode
  for (;;) {
      OneInstruction(instr); //  將指令傳入 Mipssim OneInstruction()執行
      …
  }
}
```

machine/Mipssim.cc

```
void Machine::OneInstruction(Instruction *instr)
{
    …
    case OP_SYSCALL:
        RaiseException(SyscallException, 0);
    //將 SyscallException 資訊傳入 machine RaiseException()
        return;
    …
}
```

machine/machine.h

```
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); //  從 UserMode  轉為  KernelMode
  ExceptionHandler(which);          // interrupts are enabled at this point (處裡 system call)
  kernel->interrupt->setStatus(UserMode); //  從 KernelMode  轉為  UserMode
}
```

userprog/exception.cc

```
void ExceptionHandler(ExceptionType which) //可處理 system calls 或是其他 exception
{
  int type = kernel->machine->ReadRegister(2); //將  reg2  的值取出(要處裡的 system call)
  int val;
  int status, exit, threadID, programID;
```

```
        DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
        switch (which) { //判斷是否為 system call
        case SyscallException:
            switch(type) { //判斷是哪一種  system call
            case SC_Halt:
                DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
                SysHalt();     //呼叫 ksyscall.h 的 SysHalt()
                            cout<<"in exception\n";
                ASSERTNOTREACHED();
                break;
          …
          }
    }
```

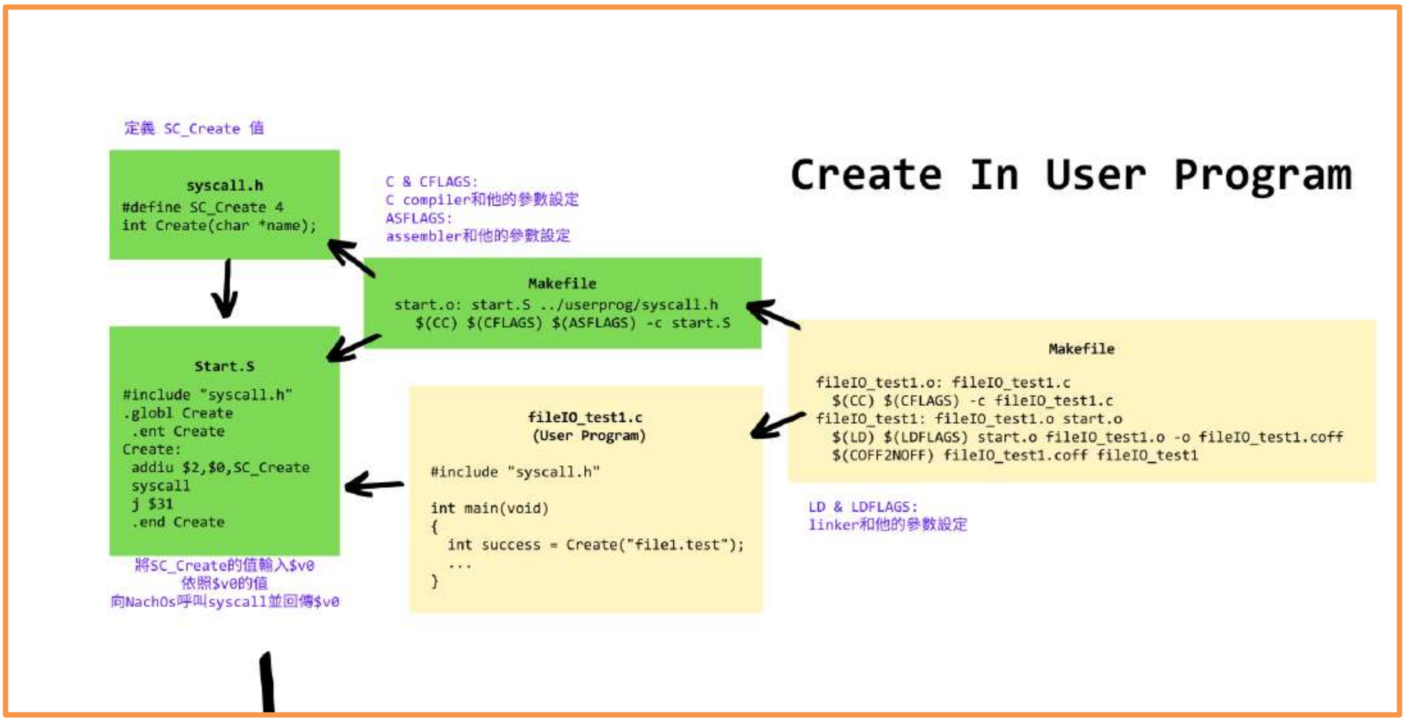
userprog/ksyscall.h

```
void SysHalt()
{
  kernel->interrupt->Halt(); //呼叫 interrupt.cc 的 Halt()
}
```

machine/interrupt.cc

```
void Interrupt::Halt() //刪除 kernel 結束程式
{
    cout << "Machine halting!\n\n";
    cout << "This is halt\n";
    kernel->stats->Print();
    delete kernel;    // Never returns.
}
```

3. Flow Chart of Create() System Call:



syscall.h

```
#define SC_Create 4
int Create(char *name);
```

Makefile

```
start.o: start.S ../userprog/syscall.h
  $(CC) $(CFLAGS) $(ASFLAGS) -c start.S
```

Makefile

```
fileIO_test1.o: fileIO_test1.c
  $(CC) $(CFLAGS) -c fileIO_test1.c
fileIO_test1: fileIO_test1.o start.o
  $(LD) $(LDFLAGS) start.o fileIO_test1.o -o fileIO_test1.coff
  $(COFF2NOFF) fileIO_test1.coff fileIO_test1
```

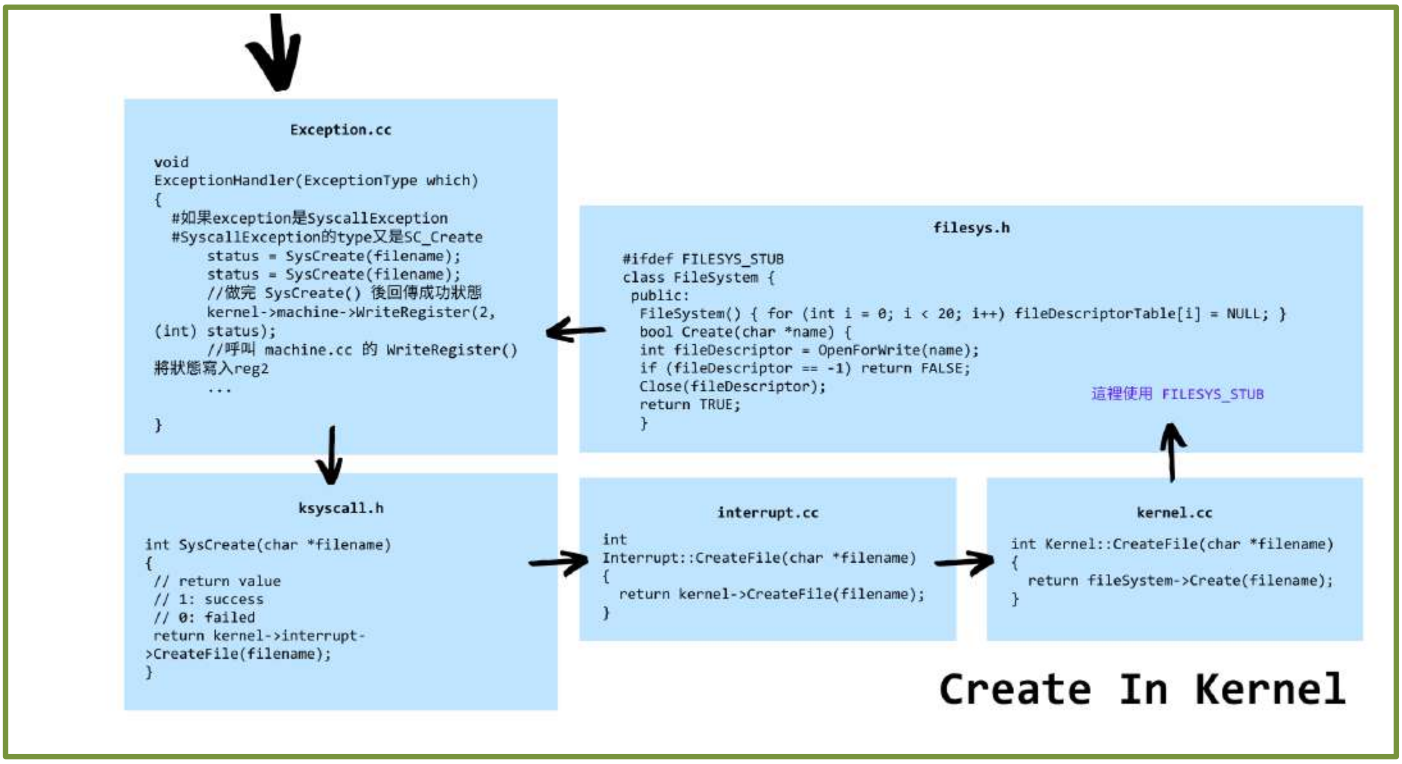
Start.S

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

fileIO_test1.c
(User Program)

```
#include "syscall.h"

int main(void)
{
  int success = Create("file1.test");
  ...
}
```



Exception.cc

```
void
ExceptionHandler(ExceptionType which)
{
  #如果exception是SyscallException
  #SyscallException的type又是SC_Create
      status = SysCreate(filename);
      status = SysCreate(filename);
      //做完 SysCreate() 後回傳成功狀態
      kernel->machine->WriteRegister(2,
(int) status);
      //呼叫 machine.cc 的 WriteRegister()
將狀態寫入reg2
      ...

}
```

filesys.h

```
#ifdef FILESYS_STUB
class FileSystem {
 public:
  FileSystem() { for (int i = 0; i < 20; i++) fileDescriptorTable[i] = NULL; }
  bool Create(char *name) {
  int fileDescriptor = OpenForWrite(name);
  if (fileDescriptor == -1) return FALSE;
  Close(fileDescriptor);
  return TRUE;
  }
```

這裡使用 FILESYS_STUB

ksyscall.h

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

interrupt.cc

```
int
Interrupt::CreateFile(char *filename)
{
  return kernel->CreateFile(filename);
}
```

kernel.cc

```
int Kernel::CreateFile(char *filename)
{
  return fileSystem->Create(filename);
}
```

Create In Kernel

4. Details of Trace Create() Code

test/start.S

```
        .globl Create
        .ent  Create
Create:
        addiu $2,$0,SC_Create //將 SC_Create 值(設為 4)存入 reg2 裡
        syscall //將參數(filename)傳入 r4 後呼叫 system call  觸發  Machine Run()
        j       $31
        .end Create
```

machine/machine.h

```
// Routines callable by the Nachos kernel
      void Run();              //  觸發  Mipssim Run()
```

machine/Mipssim.cc

```
void Machine::Run()
{
      …
      kernel->interrupt->setStatus(UserMode); //  目前為 UserMode
      for (;;) {
            OneInstruction(instr); //  將指令傳入 Mipssim OneInstruction()執行
            …
      }
}
```

machine/Mipssim.cc

```
void Machine::OneInstruction(Instruction *instr)
{
       …
```

```
        case OP_SYSCALL:
            RaiseException(SyscallException, 0);
        //將 SyscallException 資訊傳入 machine RaiseException()
            return;
        …
}
```

machine/machine.h

```
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); // 從 UserMode 轉為 KernelMode
    ExceptionHandler(which);          // interrupts are enabled at this point (處裡 system call)
    kernel->interrupt->setStatus(UserMode); // 從 KernelMode 轉為 UserMode
}
```

userprog/exception.cc

```
void ExceptionHandler(ExceptionType which) //可處理 system calls 或是其他 exception
{
    int type = kernel->machine->ReadRegister(2); //將 reg2 的值取出(要處裡的 system call)
    int val;
    int status, exit, threadID, programID;
    DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
    switch (which) { //判斷是否為 system call
    case SyscallException:
        switch(type) { //判斷是哪一種 system call
          case SC_Create:
            val = kernel->machine->ReadRegister(4); //將 reg4 的值取出(filename)
            {
            char *filename = &(kernel->machine->mainMemory[val]);
            //cout << filename << endl;
            status = SysCreate(filename); //呼叫 ksyscall.h 的 SysCreate()
            …
          …
          }
}
```

userprog/ksyscall.h

```
int SysCreate(char *filename)
{
  // return value
  // 1: success
  // 0: failed
  return kernel->interrupt->CreateFile(filename); //呼叫 interrupt.cc 的 CreateFile()
}
```

machine/interrupt.cc

```
int Interrupt::CreateFile(char *filename)
{
    return kernel->CreateFile(filename); //呼叫 kernel.cc 的 CreateFile()
}
```

threads/kernel.cc

```
int Kernel::CreateFile(char *filename)
{
    return fileSystem->Create(filename); //呼叫 filesys.h 的 Create()
     // 注意在 Makefile 已定義 flag –DFILESYS_STUB，因此只須看 filesys.h
}
```

filesys/filesys.h

```
#ifdef FILESYS_STUB          // Temporarily implement file system calls as
                             // calls to UNIX, until the real file system
                             // implementation is available
class FileSystem {
  public:
    FileSystem() { for (int i = 0; i < 20; i++) fileDescriptorTable[i] = NULL; }

    bool Create(char *name) {
    int fileDescriptor = OpenForWrite(name); //呼叫 sysdep.cc 的 OpenForWrite()

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

userprog/exception.cc

```
void ExceptionHandler(ExceptionType which)
{
    …
    switch (which) {
    case SyscallException:
        switch(type) {
           case SC_Create:
             …
             status = SysCreate(filename); //做完 SysCreate() 後回傳成功狀態
             kernel->machine->WriteRegister(2, (int) status);
                 //呼叫 machine.cc 的 WriteRegister() 將狀態寫入 reg2
             }
             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;
```

```
            …
            }
}
```

machine/machine.cc

```
void Machine::WriteRegister(int num, int value)
{
    ASSERT((num >= 0) && (num < NumTotalRegs));
    registers[num] = value;
}
```

5. **Details of Makefile**

#1. 引入所有執行檔與物件檔間的依賴關係 include Makefile.dep

#2. 指定 compiler, assempler, linker，並設定其參數

```
CC = $(GCCDIR)gcc
AS = $(GCCDIR)as
LD = $(GCCDIR)ld
INCDIR =-I../userprog -I../lib
CFLAGS = -G 0 -c $(INCDIR) -B../../usr/local/nachos/lib/gcc-lib/decstation-ultrix/2.95.2/
-B../../usr/local/nachos/decstation-ultrix/bin/
```

#3. 依實體機器的 os 決定 hosttype (如未指定 hosttype, echo 錯誤訊息) 並設定欲執行的 program

```
ifeq ($(hosttype),unknown)
  PROGRAMS = unknownhost
else # change this if you create a new test program!
  PROGRAMS = halt consoleIO_test1 consoleIO_test2 fileIO_test1 fileIO_test2 endif
```

#4. 使用 compiler 和 assemblem 編譯 start.S 使 system call 可以成功向 kernel 呼叫

```
all:
  $(PROGRAMS)
start.o: start.S ../userprog/syscall.h
  $(CC) $(CFLAGS) $(ASFLAGS) -c start.S
```

#5. (以 halt 為例)呼叫 compiler 編譯 user program，使其成為物件檔
並與其他物件檔使用 linker 連接(因中途需呼叫 sys call 故必須引入 start.o) 成為 executable file

```
halt.o: halt.c
  $(CC) $(CFLAGS) -c halt.c
halt: halt.o start.o
  $(LD) $(LDFLAGS) start.o halt.o -o halt.coff
  $(COFF2NOFF) halt.coff halt
```

可用 distclean 來移出所有 make 的物件檔和執行檔

```
clean:
  $(RM) -f *.o *.ii
  $(RM) -f *.coff
distclean: clean
  $(RM) -f $(PROGRAMS)
```

Part 2:Implement System Call

1. Detail of your Console I/O system call implementation

首先定義 SC_Print 為 16 (syscall 的值)

在 userprog/syscall.h 裡新增

```
#define SC_Print      16
void PrintInt(int num);
```

接著在 userprog/exception.cc 裡新增

```
case SyscallException:
        switch(type) {
          case SC_Print:
            val=kernel->machine->ReadRegister(4); //將要 print 的值從 reg4 拿出
            SysPrintInt(val); //將要 print 的值傳入 ksyscall.h 的 SysPrintInt()
            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);
                // 這三行為 PC+4
            return;
            ASSERTNOTREACHED();
            break;
          }
```

接著在 userprog/ksyscall.h 裡新增

```
void SysPrintInt (int number)
{
    kernel->synchConsoleOut->PutInt(number); //將值傳入 synchconsole.cc 的 PutInt()
}
```

接著在 userprog/synchconsole.cc 裡新增

```
Void SynchConsoleOutput::PutInt(int value)
{
    char str[30];
    int idx=0;
    int len = sprintf(str, "%d\n", value); // int 轉為 str
    lock->Acquire(); // 鎖定物件，開始執行同步化
    consoleOutput->PutString(str, len); //將字串及長度傳入 console.cc 的 PutString()
    waitFor->P();
    lock->Release(); // 執行完同步化，解除鎖定
}
```

另外記得在標頭檔 userprog/synchconsole.h 新增

```
class SynchConsoleOutput : public CallBackObj {
  public:
    …
    void PutInt(int value); //PutInt 函式宣告
      …
}
```

最後在 machine\console.cc 中新增

```
void ConsoleOutput::PutString(char* str, int numchar)
{
    ASSERT(putBusy == FALSE);
    WriteFile(writeFileNo, str, numchar*sizeof(char)); //寫入大小(byte)為字串大小(char 為 1byte)
    putBusy = TRUE;
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}
```

並在 machine/console.h 中新增

```
void PutString(char* str, int numchar);
```

2. Detail of your File I/O system call implementation

已在 userprog/syscall.h 裡定義 Open, Write, Read, Close 的 syscall 值：

```
#define SC_Open       6
#define SC_Read       7
#define SC_Write      8
#define SC_Close      10
OpenFileId Open(char *name);
int Write(char *buffer, int size, OpenFileId id);
int Read(char *buffer, int size, OpenFileId id);
int Close(OpenFileId id);
```

接著在 userprog/exception.cc 裡新增

```
    case SC_Open: //實作 Open
        val = kernel->machine->ReadRegister(4);
        {
        char *Openfilename = &(kernel->machine->mainMemory[val]);
        status = SysOpen(Openfilename); //將要開啟的 filename 傳入 ksyscall.h 的 SysOpen()
        kernel->machine->WriteRegister(2, (int) status);
        }
        …
        break;
     case SC_Read: //實作 Read
        val = kernel->machine->ReadRegister(4);
        size = kernel->machine->ReadRegister(5);
        id = kernel->machine->ReadRegister(6);
        {
        char *Readbuffer = &(kernel->machine->mainMemory[val]);
        status = SysRead(Readbuffer, size, id);
        //將要讀的 buffer、讀的大小及 file id 傳入 ksyscall.h 的 SysRead()處裡
        kernel->machine->WriteRegister(2, (int) status);
        }
        …
        break;
     case SC_Write: //實作 Write
        val = kernel->machine->ReadRegister(4);
        size = kernel->machine->ReadRegister(5);
```

```
            id = kernel->machine->ReadRegister(6);
            {
            char *Writebuffer = &(kernel->machine->mainMemory[val]);
            status = SysWrite(Writebuffer, size, id);
            //將要寫的 buffer、寫的大小及 file id 傳入 ksyscall.h 的 SysWrite()處裡
            kernel->machine->WriteRegister(2, (int) status);
            }
            …
            break;
        case SC_Close: //實作  Close
            id = kernel->machine->ReadRegister(4);

            status = SysClose(id); //將要關閉的 file id  傳入 ksyscall.h 的 SysClose()執行
            kernel->machine->WriteRegister(2, (int) status);

            …
            break;
```

接著在 userprog/ksyscall.h 裡新增

```
OpenFileId SysOpen(char *name) {
    // Open a file with the name, and returns its corresponding OpenFileId.
    // Return -1 if open fails
    return kernel->fileSystem->OpenF(name); //將檔名傳入 filesys.h 的 OpenF()並回傳 id

}

int SysWrite(char *buffer, int size, OpenFileId id) {
    // Write "size" characters from buffer into the file
    // Returns number of characters actually written to the file
    // If attempt writing to an invalid id, return -1
    return kernel->fileSystem->WriteF(buffer, size, id);
       //將值傳入 filesys.h 的 WriteF()並回傳實際寫的大小
}

int SysRead(char *buffer, int size, OpenFileId id) {
    // Read "size" characters from file into the buffer
    // Returns number of characters actually read from the file
    // If attempt reading from an invalid id, return -1
    return kernel->fileSystem->ReadF(buffer, size, id);
       //將值傳入 filesys.h 的 ReadF()並回傳實際讀的大小
}

int SysClose(OpenFileId id) {
    // Close the file with id
    // Return 1 if successfully close the file, 0 otherwise
    return kernel->fileSystem->CloseF(id);
       //將值傳入 filesys.h 的 CloseF()並回傳成功與否
}
```

接著在 filesys/filesys.h 裡新增

```
typedef int OpenFileId;
class FileSystem {
  public:
     OpenFileId OpenF(char *name) //實作 Open
    {
        int fileDescriptor = OpenForReadWrite(name, FALSE); //使用 lib/sysdep.cc 的函式
          if(fileDescriptor>=26) return -1; //若檔案數大於 20 回傳-1
        return fileDescriptor;
    }

    int WriteF(char *buffer, int size, OpenFileId id){ //實作 Write
      return WritePartial(id, buffer, size); //使用 lib/sysdep.cc 新增的函式
    }

    int ReadF(char *buffer, int size, OpenFileId id){ //實作 Read
      return ReadPartial(id, buffer, size); //使用 lib/sysdep.cc 的函式
    }

    int CloseF(OpenFileId id){ //實作 Close
      return (ClosePartial(id)==0); //使用 lib/sysdep.cc 新增的函式
    }
```

因為 lib/sysdep.cc 中的 Write, Read, Close 函式並不會 return 值，因此這邊使用 ReadPartial()及自定義以下函式：

```
int
WritePartial(int fd, char *buffer, int nBytes)
{
    return write(fd, buffer, nBytes); // success: 讀入數, fail: -1
}

int
ClosePartial(int fd)
{
    return close(fd);
}
```

並且在標頭檔 lib/sysdep.h 新增

```
extern int WritePartial(int fd, char *buffer, int nBytes); // 新增
extern int ClosePartial(int fd); // 新增
```

Part 3:Contribution

1. **Describe details and percentage of each member’s contribution.**

姓名	負責項目	貢獻度
何翊華	Trace code、Halt flow chart&detail、Create flow chart&detail、Implement code + detail	50%

廖偉宸	Trace code、Halt flow chart、Create flow chart、Makefile detail、Implement code + detail	50%