MP1_report_109000109

1. Trace code – SC_Halt
   A. Machine::Run( )

      當 Execute 時，系統會將 Instruction Name 吃進 Execute( )

      ```
      kernel->machine->Run();
      ```

      並觸發 Run( )

      ```
      OneInstruction(instr);
      ```

      接著觸發 Run( ) 的函式裡的 OneInstruction( )

   B. Machine::OneInstruction( )

      OneInstruction( ) 裡是執行逐條指令的過程，當發現 Syscall 時會觸發 RaiseException( )

      ```
      RaiseException(OverflowException, 0);
      ```

   C. Machine::RaiseException( )

      RaiseException( ) 將 Syscall 傳入 ExceptionHandler( ) 中

      ```
      ExceptionHandler(which);
      ```

   D. ExceptionHandler( )

      ExceptionHandler( ) 會根據 Syscall 的類型判斷該執行的事，而 SC_Halt 則在 SysHalt( ) 中執行。

   E. SysHalt( )

```
case SC_MSG:
        DEBUG(dbgSys, "Message received.\n");
        val = kernel->machine->ReadRegister(4);
        {
          char *msg = &(kernel->machine->mainMemory[val]);
          cout << msg << endl;
        }
```

```
        SysHalt();
        ASSERTNOTREACHED();
        break;
```

F. Interrupt::Halt( )

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

在 Halt( ) 中可以發現 kernel 被 delete。

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

2. Trace code – SC_Create

A. ExceptionHandler( )

```
case SC_Create:
        val = kernel->machine->ReadRegister(4);
        {
          char *filename = &(kernel->machine->mainMemory[val]);
          // cout << filename << endl;
          status = SysCreate(filename);
          kernel->machine->WriteRegister(2, (int)status);
        }
        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;
```

判斷傳入的 Exception Type ，並在 SysCreate( ) 處理 SC_Create 的相關內容。

B. SysCreate( )

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

Create( ) 在 filesystem 裡已經有 define

C. FileSystem::Create( )

```
FileSystem::Create(char *name, int initialSize)
{
    Directory *directory;
    PersistentBitmap *freeMap;
    FileHeader *hdr;
    int sector;
    bool success;

    DEBUG(dbgFile, "Creating file " << name << " size " << initialSize);

    directory = new Directory(NumDirEntries);
    directory->FetchFrom(directoryFile);

    if (directory->Find(name) != -1)
      success = FALSE;        // file is already in directory
    else {
        freeMap = new PersistentBitmap(freeMapFile,NumSectors);
        sector = freeMap->FindAndSet(); // find a sector to hold the file
header
        if (sector == -1)
```

```
            success = FALSE;       // no free block for file header
        else if (!directory->Add(name, sector))
            success = FALSE;    // no space in directory
    else {
            hdr = new FileHeader;
        if (!hdr->Allocate(freeMap, initialSize))
                success = FALSE;    // no space on disk for data
        else {
            success = TRUE;
        // everthing worked, flush all changes back to disk
                hdr->WriteBack(sector);
                directory->WriteBack(directoryFile);
                freeMap->WriteBack(freeMapFile);
        }
            delete hdr;
    }
        delete freeMap;
    }
    delete directory;
    return success;
}
```

3. Trace code – SC_PrintInt

   A. ExceptionHandler( )

首先看到的一樣是 ExceptionHandler( ) 裡的：

```
case SC_PrintInt:
        DEBUG(dbgSys, "Print Int\n");
        val = kernel->machine->ReadRegister(4);
        DEBUG(dbgTraCode, "In ExceptionHandler(), into SysPrintInt, " <<
kernel->stats->totalTicks);
        SysPrintInt(val);
        DEBUG(dbgTraCode, "In ExceptionHandler(), return from SysPrintInt, "
<< kernel->stats->totalTicks);
        // Set Program Counter
        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;
```

SysPrintInt( ) 會進入具體執行 SC_PrintInt 的地方。

B. SysPrintInt( )

```
void SysPrintInt(int val)
{
  DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, into synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
  kernel->synchConsoleOut->PutInt(val);
  DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, return from synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
}
```

進入 SynchConsoleOutput::PutInt()

C. SynchConsoleOutput::PutInt()

```
SynchConsoleOutput::PutInt(int value)
{
    char str[15];
    int idx=0;
    //sprintf(str, "%d\n\0", value);  the true one
    sprintf(str, "%d\n\0", value); //simply for trace code
    lock->Acquire();
    do{
    DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into consoleOutput->PutChar, " << kernel->stats->totalTicks);
        consoleOutput->PutChar(str[idx]);
    DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return from consoleOutput->PutChar, " << kernel->stats->totalTicks);
    idx++;
```

```
    DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into waitFor-
>P(), " << kernel->stats->totalTicks);
        waitFor->P();
    DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return form
waitFor->P(), " << kernel->stats->totalTicks);
    } while (str[idx] != '\0');
    lock->Release();
}
```

將 Syscall 參數 儲存至 str，並且使用 Lock 機制，一次只會有一個物件被鎖定。

D. SynchConsoleOutput::PutChar( )

```
void
SynchConsoleOutput::PutChar(char ch)
{
    lock->Acquire();
    consoleOutput->PutChar(ch);
    waitFor->P();
    lock->Release();
}
```

E. ConsoleOutput::PutChar( )

```
void
ConsoleOutput::PutChar(char ch)
{
    ASSERT(putBusy == FALSE);
    WriteFile(writeFileNo, &ch, sizeof(char));
    putBusy = TRUE;
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}
```

和 Int 一樣，但參數是 char。

F. Interrupt::Schedule( )

```
void Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type)
{
```

```
  int when = kernel->stats->totalTicks + fromNow;
  PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type);

  DEBUG(dbgInt, "Scheduling interrupt handler the " <<
intTypeNames[type] << " at time = " << when);
  ASSERT(fromNow > 0);

  pending->Insert(toOccur);
}
```

To call 是 interrupt 將要執行的功能或對象。

From now 是指一段時間。

G. Machine:Run( )

執行完 OneInstruction( ) 之後會進到 OneTick( )。

H. Interrupt::OneTick( )

讓系統前往下一個 Tick，像是 clock 的機制，能讓系統有中斷點。

I. Interrupt::CheckIfDue( )

檢察系統是否有如期執行 instruction，當所有 interrupt 執行結束，return true。

J. ConsoleOutput::CallBack( )

```
void
ConsoleOutput::CallBack()
{
    DEBUG(dbgTraCode, "In ConsoleOutput::CallBack(), " << kernel->stats-
>totalTicks);
    putBusy = FALSE;
    kernel->stats->numConsoleCharsWritten++;
    callWhenDone->CallBack();
}
```

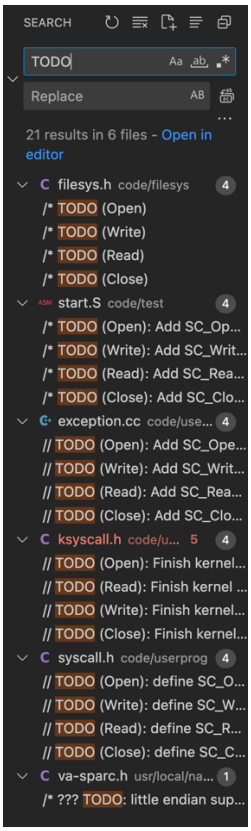
當一個字元輸出時，會呼叫此函式。

K. SynchConsoleOutput:: CallBack( )

```
void
SynchConsoleOutput::CallBack()
{
    DEBUG(dbgTraCode, "In SynchConsoleOutput::CallBack(), " << kernel-
>stats->totalTicks);
    waitFor->V();
}
```

調用 interrupt 顯示下一個字元。

4. Implement four I/O system calls in NachOS



需要更改的有這些 file ：

A. test/start.S

```
/* TODO (Open): Add SC_Open system call stubs, you can imitate existing system calls. */
    .globl Open
    .ent    Open
Open:
    addiu $2,$0,SC_Open
    syscall
    j   $31
    .end Open

/* TODO (Write): Add SC_Write system call stubs, you can imitate existing system calls. */
    .globl Write
    .ent    Write
Write:
    addiu $2,$0,SC_Open
    syscall
    j   $31
    .end Write

/* TODO (Read): Add SC_Read system call stubs, you can imitate existing system calls. */
    .globl Read
    .ent    Read
Read:
    addiu $2,$0,SC_Read
    syscall
    j   $31
    .end Read
```

借鏡 add 的寫法，starts.s 會從 $2 裡面讀取暫存器的值，並呼叫對應的 syscall ，而 Register 4, 5, 6, 7 則依序儲存 4 個參數分別是($a0, $a1, $a2, $a3)

B. userprog/syscall.h

把 define 取消註解

C. userprog/exception.cc

有四個部分需要實作，以 open 為例，

val = kernel->machine->ReadRegister(4); //取出$4 的值

char *filename = &(kernel->machine->mainMemory[val]); // 讓記憶體儲存 filename 的 pointer

status = SysOpen(filename); // 回傳執行狀況

D. userprog/ksyscall.h

間接呼叫，模擬對 kernel 進行操作。

```
// TODO (Open): Finish kernel interface for system call (Open).
OpenFileId SysOpen(char *name) {
  return kernel->fileSystem->OpenAFile(name);
}

// TODO (Read): Finish kernel interface for system call (Read).
int SysRead(char *buffer, int size, OpenFileId id) {
  return kernel->fileSystem->ReadFile(buffer, size, id);
}


// TODO (Write): Finish kernel interface for system call (Write).
int SysWrite(char *buffer, int size, OpenFileId id) {
  return kernel->fileSystem->WriteAFile(buffer, size, id);
}

// TODO (Close): Finish kernel interface for system call (Close).
int SysClose(OpenFileId id) {
  return kernel->fileSystem->CloseFile(id);
}

#endif /* ! __USERPROG_KSYSCALL_H__ */
```

E. filesys/filesys.h

```
71    OpenFileId OpenAFile(char *name) {
72      int fileDescriptor = OpenForReadWrite(name, FALSE);
73      if (fileDescriptor == -1)
74        return -1;
75      OpenFile *newFile = new OpenFile(fileDescriptor);
76
77      for (int i = 0; i < 20; i++)
78      {
79        if (OpenFileTable[i] == NULL)
80        {
81          OpenFileTable[i] = newFile;
82          return i + 1;
83        }
84      }
85      return -1;
86    }
```

- **Must handle invalid file open requests, including the non-existent file, exceeding opened file limit (at most 20 files)**

5. Test
   A. Make

```
[os23s24@localhost build.linux]$ make
g++ -g -Wall -I../network -I../filesys -I../userprog -I../threads -I../machine -I../lib -I- -DFILESYS_STUB -DRDATA -DSIM_FIX -Dx86 -DLINUX -DCHANG
ED -m32 -c ../lib/bitmap.cc
cc1plus: note: obsolete option -I- used, please use -iquote instead
g++ -g -Wall -I../network -I../filesys -I../userprog -I../threads -I../machine -I../lib -I- -DFILESYS_STUB -DRDATA -DSIM_FIX -Dx86 -DLINUX -DCHANG
ED -m32 -c ../lib/debug.cc
cc1plus: note: obsolete option -I- used, please use -iquote instead
g++ -g -Wall -I../network -I../filesys -I../userprog -I../threads -I../machine -I../lib -I- -DFILESYS_STUB -DRDATA -DSIM_FIX -Dx86 -DLINUX -DCHANG
ED -m32 -c ../lib/libtest.cc
cc1plus: note: obsolete option -I- used, please use -iquote instead
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
../lib/libtest.cc:59: warning: deprecated conversion from string constant to 'char*'
g++ -g -Wall -I../network -I../filesys -I../userprog -I../threads -I../machine -I../lib -I- -DFILESYS_STUB -DRDATA -DSIM_FIX -Dx86 -DLINUX -DCHANG
ED -m32 -c ../lib/sysdep.cc
cc1plus: note: obsolete option -I- used, please use -iquote instead
g++ -g -Wall -I../network -I../filesys -I../userprog -I../threads -I../machine -I../lib -I- -DFILESYS_STUB -DRDATA -DSIM_FIX -Dx86 -DLINUX -DCHANG
ED -m32 -c ../machine/interrupt.cc
cc1plus: note: obsolete option -I- used, please use -iquote instead
../machine/interrupt.cc:30: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:30: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:33: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:33: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:33: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:33: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:33: warning: deprecated conversion from string constant to 'char*'
../machine/interrupt.cc:33: warning: deprecated conversion from string constant to 'char*'
```

B. Teat1

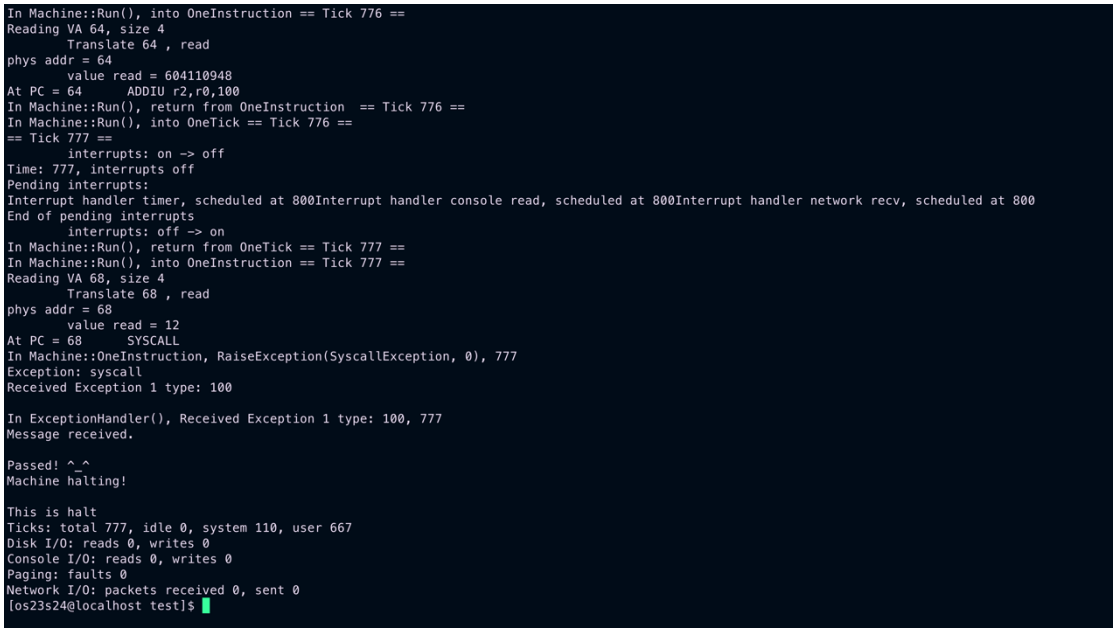
```
In Machine::Run(), into OneInstruction == Tick 924 ==
Reading VA 64, size 4
        Translate 64 , read
phys addr = 64
        value read = 604110948
At PC = 64      ADDIU r2,r0,100
In Machine::Run(), return from OneInstruction  == Tick 924 ==
In Machine::Run(), into OneTick == Tick 924 ==
== Tick 925 ==
        interrupts: on -> off
Time: 925, interrupts off
Pending interrupts:
Interrupt handler timer, scheduled at 1000Interrupt handler console read, scheduled at 1000Interrupt handler network recv, scheduled at 1000
End of pending interrupts
        interrupts: off -> on
In Machine::Run(), return from OneTick == Tick 925 ==
In Machine::Run(), into OneInstruction == Tick 925 ==
Reading VA 68, size 4
        Translate 68 , read
phys addr = 68
        value read = 12
At PC = 68      SYSCALL
In Machine::OneInstruction, RaiseException(SyscallException, 0), 925
Exception: syscall
Received Exception 1 type: 100

In ExceptionHandler(), Received Exception 1 type: 100, 925
Message received.

Success on creating file1.test
Machine halting!

This is halt
Ticks: total 925, idle 0, system 130, user 795
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
[os23s24@localhost test]$
```

C. Teat2



6. 心得：

NachOS 的 code 真的不少，需要花蠻多時間看完，但是並沒有想像中難，他使用 c 模擬出整個 OS 的環境真的很神奇，也讓我對整個系統和各個系統組成都更有概念，另外我覺得比較困難的是透過 vim 編輯，真的容易出錯，也不好發現錯誤。