CS342301: Operating System
MP1: System Call

- **Team member & contribution:**
  - I. **江佩霖 111062118**
    - i. Trace code (a), (c)
    - ii. Implement function: Open, Write, Read, Close
    - iii. 測試, Debug
  - II. **陳庭竣 111020025**
    - i. Trace code (a), (b)
    - ii. Implement function: Open, Write, Read, Close
    - iii. 測試, Debug

CS342301: Operating System

I. Trace Code

1. SC_Halt

i. Start.s

```
123  halt.o: halt.c
124    $(CC) $(CFLAGS) -c halt.c
125  halt: halt.o start.o
126    $(LD) $(LDFLAGS) start.o halt.o -o
       halt.coff
127    $(COFF2NOFF) halt.coff halt
```

```
45    .globl Halt
46    .ent  Halt
47  Halt:
48    addiu $2,$0,SC_Halt
49    syscall
50    j $31
51    .end Halt
```

makefile 中定義了 halt 的最終執行緒將來自 start.S 以及 halt.c 編譯後的文件(start.o , halt.o)，trace code 從 start.S 切入。

其中，halt 對應到的 code 如右圖，可以看到 Halt 將 SC_Halt (在 syscall.h 中定義為 0) 存到 register 2 號，並執行 syscall 指令。

ii. Machine::Run()

```
1   void Machine::Run()
2   {
3       Instruction *instr = new Instruction;
4       if (debug->IsEnabled('m'))
5       {
6           cout << "Starting program in thread: " << kernel->currentThread->getName();
7           cout << ", at time: " << kernel->stats->totalTicks << "\n";
8       }
9       kernel->interrupt->setStatus(UserMode);
10      for (;;)
11      {
12          DEBUG(dbgTraCode, "In Machine::Run(), into OneInstruction "
13                            << "== Tick " << kernel->stats->totalTicks << " ==");
14          OneInstruction(instr);
15          DEBUG(dbgTraCode, "In Machine::Run(), return from OneInstruction  "
16                            << "== Tick " << kernel->stats->totalTicks << " ==");
17          DEBUG(dbgTraCode, "In Machine::Run(), into OneTick "
18                            << "== Tick " << kernel->stats->totalTicks << " ==");
19          kernel->interrupt->OneTick();
20          DEBUG(dbgTraCode, "In Machine::Run(), return from OneTick "
21                            << "== Tick " << kernel->stats->totalTicks << " ==");
22          if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
23              Debugger();
24      }
```

Machine::Run()會呼叫 **OneInstruction()**處理 user program 解碼後的 instruction。

然後進到 interrupt 的 OneTick()函數中(主要模擬系統時間的演進，詳細說明在 SC_PrintInt 中)。

iii. Machine::OneInstruction()

```
147  void Machine::OneInstruction(Instruction *instr) {
699          //! when the instruction call the system code, will bring the code here to deal with the exception
700          case OP_SYSCALL:
701              DEBUG(dbgTraCode, "In Machine::OneInstruction, RaiseException(SyscallException, 0), " <<
                 kernel->stats->totalTicks);
702              RaiseException(SyscallException, 0);
703              return;
```

OneInstruction()是一個負責處理對應 assembly code 操作的 function。

當 halt 的 assembly code 執行到"syscall"時，OneInstruction()會跳到這個 case，觸發 **RaiseException()**。

➢ RaiseException() 傳入的第一個參數種類(定義在 machine/machine.h)

```
43  enum ExceptionType { NoException,           // Everything ok!
44                       SyscallException,      // A program executed a system call.
45                       PageFaultException,    // No valid translation found
46                       ReadOnlyException,     // Write attempted to page marked
47                                              // "read-only"
48                       BusErrorException,     // Translation resulted in an
49                                              // invalid physical address
50                       AddressErrorException, // Unaligned reference or one that
51                                              // was beyond the end of the
52                                              // address space
53                       OverflowException,     // Integer overflow in add or sub.
54                       IllegalInstrException, // Unimplemented or reserved instr.
55
56                       NumExceptionTypes
57  };
```

iv. Machine::RaiseException()

```
97  void Machine::RaiseException(ExceptionType which, int badVAddr) {
98      DEBUG(dbgMach, "Exception: " << exceptionNames[which]);
99      registers[BadVAddrReg] = badVAddr;
100     DelayedLoad(0, 0);  // finish anything in progress
101     kernel->interrupt->setStatus(SystemMode);
102     ExceptionHandler(which);  // interrupts are enabled at this point
103     kernel->interrupt->setStatus(UserMode);
104 }
```

它將 status 設成 SystemMode 並呼叫 **ExceptionHandler()**處理對應的 System Call。處理完後再將 status 重新設定回 UserMode。

v. ExceptionHandler()

```
51  void ExceptionHandler(ExceptionType which) {
52      char ch;
53      int val;
54      int type = kernel->machine->ReadRegister(2);  // read the type of exception
55      int status, exit, threadID, programID, fileID, numChar;
```

```
    switch (which) {
        case SyscallException:
            switch (type) {
                // sc_halt
                case SC_Halt:
                    DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
                    SysHalt();
                    cout << "in exception\n";
                    ASSERTNOTREACHED();
                    break;
```

先從 Register 2 號中讀取 exception 的種類(即一開始 trace code 的 start.S 存的 SC_Halt)，根據傳入的參數 exception 錯誤。這邊對應到的 type case 是 SC_Halt，因此會呼叫 **SysHalt()**。

vi. SysHalt()

```
17  void SysHalt() {
18      // call interrupt->Halt to stop cpu, crush the simulation ans system
19      kernel->interrupt->Halt();
20  }
```

呼叫 interrupt 中定義的 **Halt()**函數

vii. Interrupt::Halt()

```
 85  class Interrupt {
104      void Halt();  // quit and print out stats
```

```
286  void Interrupt::Halt()
287  {
288  #ifndef NO_HALT_STAT
289      cout << "Machine halting!\n\n";
290      cout << "This is halt\n";
291      kernel->stats->Print();
292  #endif
293      // delete the kernel
294      delete kernel; // Never returns.
295  }
```

在 interrupt.h 中的 Interrupt class 中找到對應的函數 Halt()，到 interrupt.cc 中即可看到 function code。

因為 kernal 連結了程式以及函數，刪除 kernal 後就會將程式停止。

2. SC_Create

i. start.S

```
109   .globl Create
110   .ent  Create
111 Create:
112   addiu $2,$0,SC_Create
113   syscall
114   j $31
115   .end Create
```

同 SC_Halt，會經過 Machine::Run(), Machine::OneInstruction(), Machine::RaiseException()後呼叫 **ExceptionHandler()**。

ii. ExceptionHandler()

```
51  void ExceptionHandler(ExceptionType which) {
111                 // sc_create
112                 case SC_Create:
113                     val = kernel->machine->ReadRegister(4);
114                     // create the file (filename store in the reg)
115                     {
116                         // read the addr's data from memory(store the filename)
117                         char *filename = &(kernel->machine->mainMemory[val]);
118                         // cout << filename << endl;
119                         // call syscreate to create a file with filename->filename
120                         status = SysCreate(filename);
121                         // store the result back to reg 2
122                         kernel->machine->WriteRegister(2, (int)status);
123                     }
124                     // write the state back to reg
125                     // renew the PrevPCReg、PCReg and NextPCReg, the same as sc_printint
126                     kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
127                     kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
128                     kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
129                     return;
130                     // the same work as syscall sc_halt
131                     ASSERTNOTREACHED();
132                     break;
```

```
209 void Machine::WriteRegister(int num, int value) {
210     ASSERT((num >= 0) && (num < NumTotalRegs));
211     registers[num] = value;
212 }
213
```

同 SC_Halt，這邊對應到 SC_Create，會從 register 4 號中取出存放的 filename 地址，然後再呼叫 **SysCreate()**。接著執行 machine.cc 中的 WriteRegister()，將 SysCreate() 回傳的值存到 register 2 中。

iii. SysCreate()

```
35  int SysCreate(char *filename) {
36      // return value
37      // 1: success
38      // 0: failed
39      // call filesystem->create to create a new file
40      return kernel->fileSystem->Create(filename);
41  }
```

呼叫 **Create()**函數，並回傳一個 int 代表是否成功 create file。

iv. FileSystem::Create()

```
46  class FileSystem {
52      bool Create(char *name) {
53          int fileDescriptor = OpenForWrite(name);
54
55          if (fileDescriptor == -1)
56              return FALSE;
57          Close(fileDescriptor);
58          return TRUE;
59      }
```

```
287 int OpenForWrite(char *name) {
288     int fd = open(name, O_RDWR | O_CREAT | O_TRUNC, 0666);
289
290     ASSERT(fd >= 0);
291     return fd;
292 }
```

```
369 int Close(int fd) {
370     int retVal = close(fd);
371     ASSERT(retVal >= 0);
372     return retVal;
373 }
```

可在 filesys.h 的 FileSystem class 中找到對應的 Create()函數。

Function 中呼叫的兩個函數 OpenForWrite(), Close()則在 lib/sysdep.cc 中，分別呼叫了 open() / close() 來執行檔案的開啟和關閉。

- open() 傳入的參數簡介:

  O_RDONLY: 只讀,開啟文件 / O_WRONLY: 只寫,開啟文件 / O_RDWR: 讀、寫, 開啟文件

  0666: 更改檔案權限碼(-rw-rw-rw-)

3. SC_PrintInt

i. start.S

```
53    .globl  PrintInt
54    .ent     PrintInt
55  PrintInt:
56    addiu $2,$0,SC_PrintInt
57    syscall
58    j       $31
59    .end  PrintInt
```

同 SC_Halt，會經過 Machine::Run(), Machine::OneInstruction(), Machine::RaiseException()後呼叫 **ExceptionHandler()**。

ii. ExceptionHandler()

```
51  void ExceptionHandler(ExceptionType which) {
73                  // call the function to print
74                  case SC_PrintInt:
75                      DEBUG(dbgSys, "Print Int\n");
76                      // read the value in regester 4
77                      val = kernel->machine->ReadRegister(4);
78                      // into the sysPrintInt, and record the time it spends
79                      DEBUG(dbgTraCode, "In ExceptionHandler(), into SysPrintInt, " << kernel->stats->totalTicks);
80                      // call function to print the value
81                      SysPrintInt(val);
82                      DEBUG(dbgTraCode, "In ExceptionHandler(), return from SysPrintInt, " <<
                        kernel->stats->totalTicks);
83                      // Set Program Counter
84                      // renew the regester to the next instructon's addr.
85                      // renew the PrevPCReg、PCReg and NextPCReg
86                      kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
87                      kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
88                      kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
89                      return;
90                      ASSERTNOTREACHED();
91                      break;
```

同 SC_Halt，這邊對應到 SC_PrintInt，會從 register 4 號中取出存放的 int value，然後呼叫 **SysPrintInt()**。

iii. SysPrintInt()

```
22  void SysPrintInt(int val) {
23      // print the int to console
24      //  record the time spend
25      DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, into synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
26      // print the value out
27      kernel->synchConsoleOut->PutInt(val);
28      DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, return from synchConsoleOut->PutInt, " <<
        kernel->stats->totalTicks);
29  }
```

呼叫 sysnchCosoleOut 的 **PutInt()**。

iv. SynchConsoleOutpute::PutInt()

```
114 void SynchConsoleOutput::PutInt(int value)
116     char str[15];
117     int idx = 0;
118     // sprintf(str, "%d\n\0", value);  the true one
119     // turn the int into string and save it to the variable -> str
120     sprintf(str, "%d\n\0", value); // simply for trace code
121     lock->Acquire();
122     do
123     {
124         // record the time spend when the code get into the PutChar function
125         DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into consoleOutput->PutChar, " <<
            kernel->stats->totalTicks);
126         // output the str index by index to the console
127         consoleOutput->PutChar(str[idx]);
128         DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return from consoleOutput->PutChar, " <<
            kernel->stats->totalTicks);
129
130         // increase the index
131         idx++;
132
133         // record the time spend when the code get into the waitFor function
134         DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into waitFor->P(), " << kernel->stats->totalTicks);
135         // waiting for the system to prepare to print the other number
136         // 確保在輸出下一個字符之前，設備已經處理完上一次的輸出。
137         waitFor->P();
138         DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return form  waitFor->P(), " << kernel->stats->totalTicks);
139     } while (str[idx] != '\0'); // if the idx isn't empty
140
141     lock->Release();
```

```
80  SynchConsoleOutput::SynchConsoleOutput(char *outputFile)
81  {
82      consoleOutput = new ConsoleOutput(outputFile, this);
83      lock = new Lock("console out");
84      waitFor = new Semaphore("console out", 0);
85  }
```

先將 value 轉成字串形式存到 str 中，利用 lock 機制(定義在底下 SynchConsoleOutput()中)實現同步化輸出，僅有鎖定的執行緒進到指定的 stdout。根據 str 陣列中每個字元呼叫 **PutChar()**，waitFor -> P()是讓 str[idx]後還未進到 PutChar()的字元先等待。

v. SynchConsoleOutpute::PutChar()

```
104 void SynchConsoleOutput::PutChar(char ch)
105 {
106     // output the char
107     // lock - release is to stop other process to use console until the work done
108     lock->Acquire();
109     consoleOutput->PutChar(ch);
110     waitFor->P();
111     lock->Release();
112 }
```

呼叫 **ConsoleOutput::PutChar()**，同樣有 lock 機制和 waitFor -> P()，基本上和前一個函數的差異不大，僅輸入改為輸入一個字元。

vi. ConsoleOutput::PutChar()

```
180 void ConsoleOutput::PutChar(char ch)
181 {
182     // check the output is bussy or not
183     ASSERT(putBusy == FALSE);
184     // output the ch to console
185     WriteFile(writeFileNo, &ch, sizeof(char));
186     // set the console's output to bussy status
187     putBusy = TRUE;
188     // after the consoletime, the schedule will stop the putbussy's busy state and turn it back to idle -> can output
        the next char
189     kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
190 }
```

```
334 void WriteFile(int fd, char *buffer, int nBytes) {
335     // printf("In sysdep.cc, nBytes: %d\n", nBytes);
336     int retVal = write(fd, buffer, nBytes);
337     ASSERT(retVal == nBytes);
338 }
```

呼叫 WriteFile()(function code 如下圖)，使用 write 函數來寫入 file。將 putBusy 設為 True，這個 bool value 定義在 console.h 的 ConsoleOutput class 中，控制每個字元是否可以輸出(ConsoleOutput::PutChar()會先檢查 putBusy 是不是 FALSE)。
每輸出一個字元後 PutChar()函數就會呼叫 **Schedule()**。

- ConsoleTime 參數在 machine/stats.h 中有定義，代表系統模擬的時間(int type)，其他相關的變數如下:

```
52 const int UserTick = 1;         // advance for each user-level instruction
53 const int SystemTick = 10;      // advance each time interrupts are enabled
54 const int RotationTime = 500;   // time disk takes to rotate one sector
55 const int SeekTime = 500;       // time disk takes to seek past one track
56 const int ConsoleTime = 100;    // time to read or write one character
57 const int NetworkTime = 100;    // time to send or receive one packet
58 const int TimerTicks = 100;     // (average) time between timer interrupts
```

- ConsoleReadInt 參數在 machine/interrupt.h 中有定義，代表產生 interrupt 的硬體設備，其他相關的參數如下:

```
51 // IntType records which hardware device generated an interrupt.
52 // In Nachos, we support a hardware timer device, a disk, a console
53 // display and keyboard, and a network.
54 enum IntType { TimerInt,
55                DiskInt,
56                ConsoleWriteInt,
57                ConsoleReadInt,
58                NetworkSendInt,
59                NetworkRecvInt };
```

vii. Interrupt::Schedule()

```
void Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type)
{
    // count the stop time
    // add "fromNow" to totalticks
    int when = kernel->stats->totalTicks + fromNow;
    // create a new pendingInterrupt
    PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type);

    DEBUG(dbgInt, "Scheduling interrupt handler the " << intTypeNames[type] << " at time = " << when);
    // check if the fromNow  for future
    ASSERT(fromNow > 0);
    // insert the pending into a list
    pending->Insert(toOccur);
}
```

```
PendingInterrupt::PendingInterrupt(CallBackObj *callOnInt,
                                   int time, IntType kind)
{
    callOnInterrupt = callOnInt;
    when = time;
    type = kind;
}
```

toCall 是 interrupt 執行的對象(PutChar 傳入的參數 this)；fromNow 則是 interrupt 多久後要發生(ConsoleTime, 詳細可以參考 **vi** 的補充)，變數 when 會依照現在的 totalTicks + fromNow 決定 interrupt 會發生的時間；type 則是產生 interrupt 的硬體(ConsoleReadInt, 詳細可以參考 **vi** 的補充)。

Schedule()函數會創建新的 PendingInterrupt 物件(輪到下一個 char 可以輸出的 interrupt)，並將物件由 Insert 函數放到 pending queue 中等待處理。

- Pending list 宣告

```
Interrupt::Interrupt()
{
    level = IntOff;
    // init a list to stroe the pendings
    pending = new SortedList<PendingInterrupt *>(PendingCompare);
    // init the variables
    inHandler = FALSE;
    yieldOnReturn = FALSE;
    // systemMode
    status = SystemMode;
}
```

viii. Machine::Run()

```
void Machine::Run()
{
    Instruction *instr = new Instruction;
    if (debug->IsEnabled('m'))
    {
        cout << "Starting program in thread: " << kernel->currentThread->getName();
        cout << ", at time: " << kernel->stats->totalTicks << "\n";
    }
    kernel->interrupt->setStatus(UserMode);
    for (;;)
    {
        DEBUG(dbgTraCode, "In Machine::Run(), into OneInstruction "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        OneInstruction(instr);
        DEBUG(dbgTraCode, "In Machine::Run(), return from OneInstruction  "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        DEBUG(dbgTraCode, "In Machine::Run(), into OneTick "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        kernel->interrupt->OneTick();
        DEBUG(dbgTraCode, "In Machine::Run(), return from OneTick "
                          << "== Tick " << kernel->stats->totalTicks << " ==");
        if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
            Debugger();
    }
```

同 SC_Halt，執行完 OneInstruction()後會呼叫 **OneTick()**。

ix. Machine::OneTick()

```
173 void Interrupt::OneTick()
174 {
175     MachineStatus oldStatus = status;
176     Statistics *stats = kernel->stats;
177 
178     if (status == SystemMode)
179     {
180         stats->totalTicks += SystemTick;
181         stats->systemTicks += SystemTick;
182     }
183     else
184     {
185         stats->totalTicks += UserTick;
186         stats->userTicks += UserTick;
187     }
188     DEBUG(dbgInt, "== Tick " << stats->totalTicks << " ==");
189     // check any pending interrupts are now ready to fire
190     // temprary stop the pending work to make sure the count be stop by other inturrupt
191     ChangeLevel(IntOn, IntOff); // first, turn off interrupts
192                                 // (interrupt handlers run with
193                                 // interrupts disabled)
194     // call check if due to check pending
195     CheckIfDue(FALSE);          // check for pending interrupts
196     ChangeLevel(IntOff, IntOn); // re-enable interrupts
197 
198 
199     if (yieldOnReturn)
200     { // if the timer device handler asked
201       // for a context switch, ok to do it now
202         yieldOnReturn = FALSE;
203         // switch to systemMode
204         status = SystemMode; // yield is a kernel routine
205         kernel->currentThread->Yield();
206         status = oldStatus;
207     }
208 }
```

```
124 void Interrupt::ChangeLevel(IntStatus old, IntStatus now)
125 {
126     // renew the level member
127     level = now;
128     DEBUG(dbgInt, "\tinterrupts: " << intLevelNames[old] << " -> " << intLevelNames[now]);
129 }
```

主要目的是模擬時間的演進，從 Nachos 開始計數(totalTicks)做為系統的時間。在系統跑完一個指令(執行一次 OneInstruction())，便會呼叫 OneTick()並根據現在系統的 status 加上對應的 Ticks(詳細的數字大小可以參考 **vi** 的補充)。

另外，透過 ChangeLevel()函數設置 interrupt handler，呼叫 **CheckIfDue()**函數來檢查 interrupt 的執行狀況以及解決發生的 interrupt。最後根據當前 yieldOnReturn 的值決定是否要執行 Yield()函數。

- yieldOnReturn 主要由 isHandler 決定，如果正在執行一個 interrupt 時，isHandler 會被設置為 True，則 OneTick()中 if 內的程式碼會執行，將系統設置為 system mode 後呼叫 Yield()，釋放(Relinquis)當前 thread 然後執行下一個 thread(如果有)。

```
200 void Thread::Yield() {
201     Thread *nextThread;
202     IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
203 
204     ASSERT(this == kernel->currentThread);
205 
206     DEBUG(dbgThread, "Yielding thread: " << name);
207 
208     nextThread = kernel->scheduler->FindNextToRun();
209     if (nextThread != NULL) {
210         kernel->scheduler->ReadyToRun(this);
211         kernel->scheduler->Run(nextThread, FALSE);
212     }
213     (void)kernel->interrupt->SetLevel(oldLevel);
214 }
```

- MachineStatus(即 Nachos 當前的狀態)有 idle, kernel mode, user mode 三種(定義在 interrupt.h 中)

x. Interrupt::CheckIfDue()

```
379 bool Interrupt::CheckIfDue(bool advanceClock)
381     PendingInterrupt *next;
382     Statistics *stats = kernel->stats;
383     // check the level is off, to handle the interrupt carefully
384     ASSERT(level == IntOff); // interrupts need to be disabled,
385                              // to invoke an interrupt handler
386     if (debug->IsEnabled(dbgInt))
387     {
388         DumpState();
389     }
390     if (pending->IsEmpty())
391     { // no pending interrupts
392         return FALSE;
393     }
394     next = pending->Front();
395     if (next->when > stats->totalTicks)
396     {
397         if (!advanceClock)
398         {
399             return FALSE;
400         }
401         else
402         { // advance the clock to next interrupt
403             stats->idleTicks += (next->when - stats->totalTicks);
404             stats->totalTicks = next->when;
405             // UDelay(1000L); // rcgood - to stop nachos from spinning.
406         }
407     }
408     DEBUG(dbgInt, "Invoking interrupt handler for the ");
409     DEBUG(dbgInt, intTypeNames[next->type] << " at time " << next->when);
410     if (kernel->machine != NULL)
411     {
412         kernel->machine->DelayedLoad(0, 0);
413     }
414     inHandler = TRUE;
415     do
416     {
417         next = pending->RemoveFront(); // pull interrupt off list
418         DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, into callOnInterrupt->CallBack, " << stats->totalTicks);
419         next->callOnInterrupt->CallBack(); // call the interrupt handler
420         DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, return from callOnInterrupt->CallBack, " << stats->totalTicks);
421         delete next;
422     } while (!pending->IsEmpty() && (pending->Front()->when <= stats->totalTicks));
423     inHandler = FALSE; // all done
424     return TRUE;
```

確認到現在時間的 interrupt 是否有如期的發生並被解決。

若當前有需要執行的 interrupt，會進到 do-while 迴圈中，並呼叫 **CallBack()**。

若直到下一個 interrupt 前沒有任何事件(根據 advanceClock 的值判斷)，則將時間快進到下一個 interrupt 發生的時間點。

xi. ConsoleOutput::CallBack()

```
162 void ConsoleOutput::CallBack()
163 {
164     // call the console to output the next char
165     DEBUG(dbgTraCode, "In ConsoleOutput::CallBack(), " << kernel->stats->totalTicks);
166     // set the bussy state back to idle
167     putBusy = FALSE;
168     // record the number of output char
169     kernel->stats->numConsoleCharsWritten++;
170     // wake up the other idle output process to start output the file they want
171     callWhenDone->CallBack();
172 }
```

當在 Schedule()函數中放到 pending list 的物件被觸發並處理(下個字元可以輸出到顯示器)，模擬器會調用這個函數，將 putBusy 設置為 FALSE。並呼叫 **SynchConsoleOutput::CallBack()**。

xii. SynchConsoleOutput::CallBack()

```
159 void SynchConsoleOutput::CallBack()
160 {
161     DEBUG(dbgTraCode, "In SynchConsoleOutput::CallBack(), " << kernel->stats->totalTicks);
162     waitFor->V();
163 }
```

當可以安全地發送下一個字元時，由 interrupt handler 呼叫，設置 waitFor -> V()，調用 interrupt 並送到顯示器。

4. **Makefile**(分成三個部分)

```
include Makefile.dep

CC = $(GCCDIR)gcc
AS = $(GCCDIR)as
LD = $(GCCDIR)ld

INCDIR =-I../userprog -I../lib
CFLAGS = -g -G 0 -c $(INCDIR) -B/usr/bin/local/nachos/lib/gcc-lib/decstation-ultrix/2.95.2/ -B/usr/bin/local/nachos/decstation-ultrix/bin/

ifeq ($(hosttype),unknown)
PROGRAMS = unknownhost
else
# change this if you create a new test program!
PROGRAMS = add halt createFile LotOfAdd
endif
```

首先是基本設置。首行讀取了 Makefile.dep 內容(主要是檢測作業系統的名稱、版本，以及設置了 Linux, Windows, MaxOS 作業系統上的相關設置，包括 CPP/GCCDIR/LDFLAGS 等等文件的目錄、路徑參數)。

接著設定 C 語言編譯器、組合語言編譯器以及 Linker 的路徑(行數 104-106)；指定在編譯時所需的標頭檔(.h 文件)的檢索路徑；以及 CFLAGS，編譯時的相關參數。

Ifeq 的部分，根據 hosttype 的值判斷要編譯哪些程式(在.dep 中預設是 unknown)，若不是 unknown 則會編譯 add、halt、createFile、LotOfAdd 這四個程式。

```
all: $(PROGRAMS)

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

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

add.o: add.c
  $(CC) $(CFLAGS) -c add.c

add: add.o start.o
  $(LD) $(LDFLAGS) start.o add.o -o add.coff
  $(COFF2NOFF) add.coff add
```

首行表示當執行 make all 時會編譯$(PROGRAMS)變數中的所有目標。

冒號前後是目標以及其依賴的源文件(“生成目標文件：源文件”)，後面會接上對應的規則以及對應(使用規則)的文件

123-124 行，指定生成 halt.o 所需要的源文件 halt.c，依照設置的編譯器以及相關的選項來編譯 halt.c 並生成 halt.o。

125-127 行，先指定生成 halt 文件需要的源文件 halt.o, start.o (makefile 會確保兩個源文件在執行 make halt 前已經編譯好才執行)，接著連結 start.o, halt.o 並生成一個可執行文件 halt.coff (-o 是指定檔案名的參數)，最後則是將.coff 檔案轉換成 NachOS 使用的 NOFF 格式，最後輸出 halt。其餘檔案的除檔名外都相同，故略。

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

distclean: clean
  $(RM) -f $(PROGRAMS)

unknownhost:
  @echo Host type could not be determined.
  @echo make is terminating.
  @echo If you are on an MFCF machine, contact the instructor to report this problem
  @echo Otherwise, edit Makefile.dep and try again.
```

當執行 make clean 時，會執行以下兩個步驟(刪除所有.o, .ii, .coff 文件)

若執行 make distclean 則會一併刪除編譯好的 programs。

最後 unknownhost 則會對於無法識別的 hosttype 輸出錯誤訊息。

II. Implement the I/O system calls in NachOS

1. OpenFileId Open(char *name);
2. int Write(char *buffer, int size, OpenFileId id);
3. int Read(char *buffer, int size, OpenFileId id);
4. int Close(OpenFileId id);

```
117  /* ----------------------------------------------------------------
120   * ----------------------------------------------------------------
121   */
122    .globl Open
123    .ent  Open
124  Open:
125    addiu $2,$0,SC_Open
126    syscall
127    j $31
128    .end Open
129
130    .globl Read
131    .ent  Read
132  Read:
133    addiu $2,$0,SC_Read
134    syscall
135    j $31
136    .end Read
137
138    .globl Write
139    .ent  Write
140  Write:
141    addiu $2,$0,SC_Write
142    syscall
143    j $31
144    .end Write
145
146    .globl Close
147    .ent  Close
148  Close:
149    addiu $2,$0,SC_Close
150    syscall
151    j $31
152    .end Close
153  /*
     ----------------------------------------------------------------
```

首先進到 start.S，將四個 function call 的 assembly code 寫上(參考 SC_Create 的寫法)

```
27  #define SC_Open 6   // task function
28  #define SC_Read 7   // task function
29  #define SC_Write 8  // task function
30  #define SC_Seek 9
31  #define SC_Close 10  // task function
```

將 syscall.h 中的 define 註解拿掉。

```
// sc_Open
case SC_Open:
    // read the addr in reg 4
    val = kernel->machine->ReadRegister(4);
    // open the file (filename store in the reg)
    {
        // read the addr's data from memory(store the filename)
        char *filename = &(kernel->machine->mainMemory[val]);
        // cout << filename << endl;
        // call SysOpen to open a file with filename->filename
        status = SysOpen(filename);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int)status);
    }
    // write the state back to reg
    // renew the PrevPCReg、PCReg and NextPCReg, the same as sc_printint
    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;
    // the same work as syscall sc_halt
    ASSERTNOTREACHED();
    break;
```

```
// SC_Write
case SC_Write:
    // read the addr in reg 4
    val = kernel->machine->ReadRegister(4);
    numChar = kernel->machine->ReadRegister(5);
    fileID = kernel->machine->ReadRegister(6);
    // open the file (filename store in the reg)
    {
        // read the addr's data from memory(store the filename)
        char *buf = &(kernel->machine->mainMemory[val]);
        // cout << filename << endl;
        // call SysOpen to open a file with filename->filename
        status = SysWrite(buf, numChar, fileID);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int)status);
    }
    // write the state back to reg
    // renew the PrevPCReg、PCReg and NextPCReg, the same as sc_printint
    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;

    // the same work as syscall sc_halt
    ASSERTNOTREACHED();
    break;
```

根據 exception handler 的註解，從 register 中分別讀取對應的資料，呼叫 **ksyscall.h** 中實現操作的函數。之後將 status 存入 2 號 register 完成操作。

```
// SC_Read
case SC_Read:
    // read the addr in reg 4
    val = kernel->machine->ReadRegister(4);
    numChar = kernel->machine->ReadRegister(5);
    fileID = kernel->machine->ReadRegister(6);
    // open the file (filename store in the reg)
    {
        // read the addr's data from memory(store the filename)
        char *buf = &(kernel->machine->mainMemory[val]);

        // cout << filename << endl;
        // call SysOpen to open a file with filename->filename
        status = SysRead(buf, numChar, fileID);
        // status = SysOpen(filename);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int)status);
    }
    // write the state back to reg
    // renew the PrevPCReg、PCReg and NextPCReg, the same as sc_printint
    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;
    // the same work as syscall sc_halt
    ASSERTNOTREACHED();
    break;
```

```
// SC_Close
case SC_Close:
    // read the addr in reg 4
    fileID = kernel->machine->ReadRegister(4);
    // open the file (filename store in the reg)
    {
        // read the addr's data from memory(store the filename)
        // char *filename = &(kernel->machine->mainMemory[val]);
        // cout << filename << endl;
        // call SysOpen to open a file with filename->filename
        status = SysClose(fileID);
        // status = Remove(filename);
        // store the result back to reg 2
        kernel->machine->WriteRegister(2, (int)status);
    }
    // write the state back to reg
    // renew the PrevPCReg、PCReg and NextPCReg, the same as sc_printint
    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;
    // the same work as syscall sc_halt
    ASSERTNOTREACHED();
    break;
// end TODO
```

(同上)

```
45  OpenFileId SysOpen(char *name) {
46      return kernel->fileSystem->OpenAFile(name);
47  }
48
49  int SysWrite(char *buffer, int size, OpenFileId id) {
50      // Change the name into WriteAFile, else the function can't call the syscall.h because the function's name are the same :)
51      return kernel->fileSystem->WriteAFile(buffer, size, id);
52  }
53
54  int SysRead(char *buffer, int size, OpenFileId id) {
55      return kernel->fileSystem->ReadFile(buffer, size, id);
56  }
57
58  int SysClose(OpenFileId id) {
59      int status = kernel->fileSystem->CloseFile(id);
60      if (status == 0)
61          return 1;
62      return 0;
63  }
```

Ksyscall.h 中的函數，分別再呼叫 **filesys.h** 的函數。其中比較需要注意的點是 SysWrite 呼叫的函數名字要更改。

```
int OpenAFile(char *name) {
    // OpenForReadWrite ->
    int fileDescriptor = OpenForReadWrite(name, FALSE);
    if (fileDescriptor == -1)
        return FALSE;
    return fileDescriptor;
}
int WriteAFile(char *buffer, int size, OpenFileId id) {
    // call writeFile
    // open for read write
    if (size <= 0) {
        return -1;
    }
    // syscall.h :)
    WriteFile(id, buffer, size);
    return size;
}
int ReadFile(char *buffer, int size, OpenFileId id) {
    if (size <= 0) {
        return -1;
    }

    Read(id, buffer, size);
    return size;
}
int CloseFile(OpenFileId id) {
    int status = Close(id);
    return status;
}
```

Filesys.h 中對應函數寫法。可直接使用 sysdep.cc 中定義好的函式來撰寫。

III. Report

1. What difficulties did you encounter when implementing this assignment?

   - 江佩霖

     在進行第二部分實際寫程式時，一開始因為內容太多會不太清楚要從哪部分下手，但後來發現其實寫這幾個 system call 的邏輯蠻大一部分與 trace code 中的內容相似，運用相同的概念，再針對每種 system call 做對應的調整就能順利完成。

   - 陳庭峻

     架設環境的過程比較多步驟要做，可能會遇到一些講義中沒有提到的狀況需要上網查詢相關的資料才能解決，這部分我們花了比較多時間。

2. Feedback

   - 江佩霖

     助教回信速度超快 XD。每次討論過後，我對整個系統的運作都有更深入的理解。實作 function 的過程也很有成就感。最大的困難點在於一開始面對龐大的 code 海，即使理解了內容，也常常因為數量太多，搞混 function 的位置或邏輯（雖然主要是因為我的大腦記憶力有限，存不下太多資訊）。每次都需要重新梳理一遍，但這也讓我對系統更加熟悉，覺得這樣的過程非常有趣！

   - 陳庭峻

     完成這次 lab 後，我對於作業系統如何運作有更實際的了解，包括如何處理 exception、從 user mode 切換到 kernel mode 的細節等，我也看到許多之前計算機結構學過的內容如何應用在作業系統上，希望在之後的 lab 中可以學習到更多實用的知識。

3. Reference

   i. [open 函數詳解與 close 函數詳解](https://blog.csdn.net/dangzhangjing97/article/details/79631173)

   ii. [c/c++ read 函數和 write 函數](https://blog.csdn.net/king16304/article/details/52192259)

   iii. [c++11 yield 函数的使用](https://blog.csdn.net/c_base_jin/article/details/79246211)