MP1_report_56

(a). Cover page, including

team member list, team member contributions

- 資工大三 108020022 周昱宏
- 資工大三 108071003 李彥璋

工作項目	負責人
Trace code	周昱宏、李彥璋
Open、Write	周昱宏
Read、Close	李彥璋
Report	周昱宏、李彥璋
測試、Debug	周昱宏、李彥璋

(b)Explain how system calls work in NachOS, as requested in Part II-1.

(a)SC_Halt

在userprog/syscalls.h中，定義了Nachos的11個system call，其中SC_Halt為[#define SC_Halt 0]。此外，syscalls.h也宣告了許多製造system call exception的C++函式，它們的實作部分都可以在test/start.S中找到。

在test/start.S中對應到SC_Halt的stub如下：

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

[48] 將SC_Halt system call type與0號register相加，並放到2號register。

[49] 調用SC_Halt。

[50] 跳轉到31號register（儲存程序的返回地址）。

關於system call register：

2號register - 存放system call類型、system call 返回值。

4、5、6、7號register - 分別存放system call第一、二、三、四個參數。

Machine::Run()

Run() 是一個無限迴圈，主要是負責呼叫mipssim.cc中的OneInstruction()，重複動作。

```
56  void
57  Machine::Run()
58  {
59      Instruction *instr = new Instruction;  // storage for decoded instruction
60
61      if (debug->IsEnabled('m')) {
62          cout << "Starting program in thread: " << kernel->currentThread->getName();
63      cout << ", at time: " << kernel->stats->totalTicks << "\n";
64      }
65      kernel->interrupt->setStatus(UserMode);
66      for (;;) {
67      DEBUG(dbgTraCode, "In Machine::Run(), into OneInstruction " << "== Tick " << kernel->stats->totalTicks << " ==");
68          OneInstruction(instr);
69      DEBUG(dbgTraCode, "In Machine::Run(), return from OneInstruction  " << "== Tick " << kernel->stats->totalTicks << " ==");
70
71      DEBUG(dbgTraCode, "In Machine::Run(), into OneTick " << "== Tick " << kernel->stats->totalTicks << " ==");
72      kernel->interrupt->OneTick();
73      DEBUG(dbgTraCode, "In Machine::Run(), return from OneTick " << "== Tick " << kernel->stats->totalTicks << " ==");
74      if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
75          Debugger();
76      }
77  }
```

[65] Kernel mode -> User mode。

[68] 將system call指令傳給mipssim.cc中的OneInstruction()函數。

[72] 指令執行完後呼叫interrupt.cc中的OneTick()函數，增加模擬時鐘的計數。

Machine::Oneinstruction()

Oneinstruction()負責實際執行instruction。它透過register取得目前instruction的位址，解碼後執行。

在這裡可以找到system call的執行方式：

```
675         case OP_SYSCALL:
676       DEBUG(dbgTraCode, "In Machine::OneInstruction, RaiseException(SyscallException, 0), " << kernel->stats->totalTicks);
677       RaiseException(SyscallException, 0);
678       return;
```

[677] 當遇到system call指令，呼叫machine.cc中的RaiseException()函數。

Machine::RaiseException()

```
100  void
101  Machine::RaiseException(ExceptionType which, int badVAddr)
102  {
103      DEBUG(dbgMach, "Exception: " << exceptionNames[which]);
104      registers[BadVAddrReg] = badVAddr;
105      DelayedLoad(0, 0);          // finish anything in progress
106      kernel->interrupt->setStatus(SystemMode);
107      ExceptionHandler(which);        // interrupts are enabled at this point
108      kernel->interrupt->setStatus(UserMode);
109  }
```

[106] user mode -> kernel mode，因為user program調用sys call或有exception發生。

[107] 將system call exception傳入exception.cc中的ExceptionHandler()函數中。

[108] ExceptionHandler()執行完畢，kernel mode -> user mode。

Userprog::ExceptionHandler()

```
50  void
51  ExceptionHandler(ExceptionType which)
52  {
53      char ch;
54      int val;
55      int type = kernel->machine->ReadRegister(2);
56      int status, exit, threadID, programID, fileID, numChar;
57      DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
58      DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which <<
59      switch (which) {
60      case SyscallException:
61      switch(type) {
62          case SC_Halt:
63          DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
64          SysHalt();
65          cout<<"in exception\n";
66          ASSERTNOTREACHED();
67          break;
```

[55] 從2號register中取出system call type。

[59] 判斷傳入函數的是system call還是異常。

[60-61] 如果是system call，則對type進行判斷system call的類型。

[62-67] type為SC_Halt。

[64] 呼叫ksyscall.h中的SysHalt()函數，具體處理SC_Halt。

Userprog::SysHalt()

```
19 void SysHalt()
20 {
21   kernel->interrupt->Halt();
22 }
```

[21] 呼叫interrupt.cc中的Halt()函數。

machine::Halt()

```
236   void
237   Interrupt::Halt()
238   {
239       cout << "Machine halting!\n\n";
240       cout << "This is halt\n";
241       kernel->stats->Print();
242       delete kernel;  // Never returns.
243   }
```

[241] 印出performance statistics。

[242] delete kernel；Shut down Nachos（程式停止）。

(b)SC_Create

Userprog::ExceptionHandler()

```
91      case SC_Create:
92      val = kernel->machine->ReadRegister(4);
93      {
94      char *filename = &(kernel->machine->mainMemory[val]);
95      //cout << filename << endl;
96      status = SysCreate(filename);
97      kernel->machine->WriteRegister(2, (int) status);
98      }
99      kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
100     kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
101     kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
102     return;
103     ASSERTNOTREACHED();
104     break;
```

[92] 從4號register取出system ccall的第一個參數。

[97] 呼叫ksyscall.h中的SysCreate()函數，處理SC_Create。

[97,99-101] WriteRegister()函數將value寫入user program register當中，用以debug。

Userprog::SysCreate()

```
36 int SysCreate(char *filename)
37 {
38   // return value
39   // 1: success
40   // 0: failed
41   return kernel->fileSystem->Create(filename);
42 }
```

[41] 呼叫filesys.h中的Create()函數。

Filesys::Create()

```
41 #ifdef FILESYS_STUB     // Temporarily implement file system calls as
42         // calls to UNIX, until the real file system
43         // implementation is available
44 typedef int OpenFileId;
45
46 class FileSystem {
47   public:
48     FileSystem() {
49   for (int i = 0; i < 20; i++) OpenFileTable[i] = NULL;
50     }
51
52     bool Create(char *name) {
53   int fileDescriptor = OpenForWrite(name);
54
55   if (fileDescriptor == -1) return FALSE;
56   Close(fileDescriptor);
57   return TRUE;
58     }
```

[41] 暫時先用已經先定義好的FILESYS_STUB。

[53] 呼叫sysdep.cc中的OpenForWrite()函數，OpenForWrite()會呼叫C原本的open()函式，開啟一個可讀寫的檔案。

(c)SC_PrintInt

ExceptionHandler()

```
50  void
51  ExceptionHandler(ExceptionType which)
52  {
53      char ch;
54      int val;
55      int type = kernel->machine->ReadRegister(2);
56      int status, exit, threadID, programID, fileID, numChar;
57      DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
58      DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which << " type: " << type << ", " << kernel->stats->totalTicks);
59      switch (which) {
60      case SyscallException:
61      switch(type) {
62          case SC_Halt:
63          DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
64          SysHalt();
65          cout<<"in exception\n";
66          ASSERTNOTREACHED();
67          break;
68          case SC_PrintInt:
69          DEBUG(dbgSys, "Print Int\n");
70          val=kernel->machine->ReadRegister(4);
71          DEBUG(dbgTraCode, "In ExceptionHandler(), into SysPrintInt, " << kernel->stats->totalTicks);
72          SysPrintInt(val);
73          DEBUG(dbgTraCode, "In ExceptionHandler(), return from SysPrintInt, " << kernel->stats->totalTicks);
74          // Set Program Counter
75          kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
76          kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
77          kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
78          return;
79          ASSERTNOTREACHED();
80          break;
```

功能：根據Exception Type以及2號Register的value決定處理Exception方式。

[60]若Exception Type是System call，則進入SyscallException的case。

[61]若此時System call的type(2號Register的value)為SC_PrintInt則進入[68]的case。

[70]自4號Register的value指定給變數val，也就是要輸出的值。

[72]以val為參數，呼叫ksyscall.h中的SysPrintInt()。

[75-77]設定下一次的program counter。

SysPrintInt()

```
24  void SysPrintInt(int val)
25  {
26    DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, into synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
27    kernel->synchConsoleOut->PutInt(val);
28    DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, return from synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
29  }
```

功能：呼叫synchconsole.cc中的SynchConsoleOutput::PutInt()函式。

[27]以ExceptionHandler()傳遞過來的val為參數，呼叫synchconsole.cc中的SynchConsoleOutput::PutInt()。

SynchConsoleOutput::PutInt()

```
109  void
110  SynchConsoleOutput::PutInt(int value)
111  {
112      char str[15];
113      int idx=0;
114      //sprintf(str, "%d\n\0", value);  the true one
115      sprintf(str, "%d\n\0", value); //simply for trace code
116      lock->Acquire();
117      do{
118      DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into consoleOutput->PutChar, " << kernel->stats->totalTicks);
119          consoleOutput->PutChar(str[idx]);
120      DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return from consoleOutput->PutChar, " << kernel->stats->totalTicks);
121      idx++;
122
123      DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into waitFor->P(), " << kernel->stats->totalTicks);
124          waitFor->P();
125      DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return form  waitFor->P(), " << kernel->stats->totalTicks);
126      } while (str[idx] != '\0');
127      lock->Release();
128  }
```

功能：將整數參數轉換成字元一一呼叫console.cc中的ConsoleOutput::PutChar()函式。

[115]自SysPrintInt()傳遞過來的整數value，格式化成字串str。

[116]lock->Acquire()確保資源此時只給此thread運行。

[117-126]使用迴圈遍歷str字串中的字元，當字元為’\0’則跳出迴圈。

[119]以str遍歷到的字元為參數，傳遞給console.cc中的ConsoleOutput::PutChar()。

[124]waitFor->P()等待此process正常中止隨之繼續。

[127]lock->Acquire()解除lock->Acquire()鎖定。

SynchConsoleOutput::PutChar()

```
100   void
101   SynchConsoleOutput::PutChar(char ch)
102   {
103       lock->Acquire();
104       consoleOutput->PutChar(ch);
105       waitFor->P();
106       lock->Release();
107   }
```

功能：與上SynchConsoleOutput::PutInt()功能相同，只差在傳遞的是整數參數(需要再進一步轉換為字串)與字元參數。

[103]lock->Acquire()確保資源此時只給此thread運行。

[104]以ch字元為參數，傳遞給console.cc中的ConsoleOutput::PutChar()。

[105]waitFor->P()等待此process正常中止隨之繼續。

[106]lock->Acquire()解除lock->Acquire()鎖定。

ConsoleOutput::PutChar()

```
167   void
168   ConsoleOutput::PutChar(char ch)
169   {
170       ASSERT(putBusy == FALSE);
171       WriteFile(writeFileNo, &ch, sizeof(char));
172       putBusy = TRUE;
173       kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
174   }
```

功能：將欲寫入字元利用interrupt.cc的Interrupt::Schedule()函式，進行預約排程。

[170]確定putBusy是false。

[171]將SynchConsoleOutput::PutInt()或SynchConsoleOutput::PutChar()傳遞過來的ch字元寫入writeFileNo。

[172]將putBusy設為true，防止其他事情一起做。

[173]將上述以約定的時間與system call類型排程給interrupt.cc的Interrupt::Schedule()函式。

Interrupt::Schedule()

```
297   void
298   Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type)
299   {
300       int when = kernel->stats->totalTicks + fromNow;
301       PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type);
302
303       DEBUG(dbgInt, "Scheduling interrupt handler the " << intTypeNames[type] << " at time = " << when);
304       ASSERT(fromNow > 0);
305
306       pending->Insert(toOccur);
307   }
```

功能：將設定好參數的PendingInterrupt物件插入排程。

[300]將傳遞過來的fromNow變數加上此時的totalTicks即是約定排成的時間。

[301]將排程物件、預約排程時間、類型利用PendingInterrupt()函式來產生PendingInterrupt物件toOccur。

[304]確定由console.cc中的ConsoleOutput::PutChar()函式傳遞過來的fromNow大於0。

[306]將toOccur列入排程。

Machine::Run()

```
56  void
57  Machine::Run()
58  {
59      Instruction *instr = new Instruction;  // storage for decoded instruction
60
61      if (debug->IsEnabled('m')) {
62          cout << "Starting program in thread: " << kernel->currentThread->getName();
63      cout << ", at time: " << kernel->stats->totalTicks << "\n";
64      }
65      kernel->interrupt->setStatus(UserMode);
66      for (;;) {
67      DEBUG(dbgTraCode, "In Machine::Run(), into OneInstruction " << "== Tick " << kernel->stats->totalTicks << " ==");
68          OneInstruction(instr);
69      DEBUG(dbgTraCode, "In Machine::Run(), return from OneInstruction  " << "== Tick " << kernel->stats->totalTicks << " ==");
70
71      DEBUG(dbgTraCode, "In Machine::Run(), into OneTick " << "== Tick " << kernel->stats->totalTicks << " ==");
72      kernel->interrupt->OneTick();
73      DEBUG(dbgTraCode, "In Machine::Run(), return from OneTick " << "== Tick " << kernel->stats->totalTicks << " ==");
74      if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
75          Debugger();
76      }
77  }
```

功能：利用無限迴圈呼叫mipssim.cc中的OneInstruction()與OneTick()。

[65] Kernel mode -> User mode。

[68] 將system call指令傳給mipssim.cc中的OneInstruction()函數。

[72] 指令執行完後呼叫interrupt.cc中的OneTick()函數，增加模擬時鐘的計數。

Interrupt::OneTick()

```
147   void
148   Interrupt::OneTick()
149   {
150       MachineStatus oldStatus = status;
151       Statistics *stats = kernel->stats;
152
153   // advance simulated time
154       if (status == SystemMode) {
155           stats->totalTicks += SystemTick;
156       stats->systemTicks += SystemTick;
157       } else {
158       stats->totalTicks += UserTick;
159       stats->userTicks += UserTick;
160       }
161       DEBUG(dbgInt, "== Tick " << stats->totalTicks << " ==");
162
163   // check any pending interrupts are now ready to fire
164       ChangeLevel(IntOn, IntOff); // first, turn off interrupts
165                                   // (interrupt handlers run with
166                                   // interrupts disabled)
167       CheckIfDue(FALSE);      // check for pending interrupts
168       ChangeLevel(IntOff, IntOn); // re-enable interrupts
169       if (yieldOnReturn) {    // if the timer device handler asked
170                               // for a context switch, ok to do it now
171       yieldOnReturn = FALSE;
172       status = SystemMode;        // yield is a kernel routine
173       kernel->currentThread->Yield();
174       status = oldStatus;
175       }
176   }
```

功能：時間計數、確定pending的interrupt是否正常發生以及context switch。

[154-160]根據不同的mode做不同的時間計數。

[164-167]利用Interrupt::CheckIfDue()來檢查pending的interrupt是否正常發生。

[169-175]這裡做context switch，若yieldOnReturn為true，則將其設成false，並釋放kernel中目前的thread，接著執行ready queue的第一個執行項目。

Interrupt::CheckIfDue()

```
355     inHandler = TRUE;
356     do {
357         next = pending->RemoveFront();    // pull interrupt off list
358         DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, into callOnInterrupt->CallBack, " << stats->totalTicks);
359         next->callOnInterrupt->CallBack();// call the interrupt handler
360         DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, return from callOnInterrupt->CallBack, " << stats->totalTicks);
361     delete next;
362     } while (!pending->IsEmpty()
363           && (pending->Front()->when <= stats->totalTicks));
364     inHandler = FALSE;
365     return TRUE;
366 }
```

功能：確定pending的interrupt是否正常發生。

[357]當pending不為空時取得pending的第一個元素指定給next變數，並呼叫next的CallBack function，回傳完即刪除他進行下一次迴圈。

[362]當pending為空，則將inHandler設成false，並回傳true。

ConsoleOutput::CallBack()

```
152 void
153 ConsoleOutput::CallBack()
154 {
155     DEBUG(dbgTraCode, "In ConsoleOutput::CallBack(), " << kernel->stats->totalTicks);
156     putBusy = FALSE;
157     kernel->stats->numConsoleCharsWritten++;
158     callWhenDone->CallBack();
159 }
```

功能：當下一個字元可以被輸出顯示的時候，就會呼叫此call back function。

SynchConsoleOutput::CallBack()

```
136 void
137 SynchConsoleOutput::CallBack()
138 {
139     DEBUG(dbgTraCode, "In SynchConsoleOutput::CallBack(), " << kernel->stats->totalTicks);
140     waitFor->V();
141 }
```

功能：當下一個字元可以安全的寄送給輸出顯示，則Interrupt handler會呼叫此call back function。

(c). Explain your implementation, as requested in Part II-2.

- 作業當中有修改的檔案：
  - ✓ test/start.S
  - ✓ userprog/syscall.h
  - ✓ userprog/exception.cc
  - ✓ userprog/ksyscall.h
  - ✓ filesys/filesys.h

以下將依序說明：

(1). userprog/syscall.h

將下面4行的註解刪掉：

1. #define SC_Open 6
2. #define SC_Read 7
3. #define SC_Write 8
4. #define SC_Close 10

(2). test/start.S

```
165     .globl Open
166     .ent    Open
167 Open:
168     addiu $2, $0, SC_Open
169     syscall
170     j   $31
171     .end Open
172
173     .globl Read
174     .ent    Read
175 Read:
176     addiu $2, $0, SC_Read
177     syscall
178     j   $31
179     .end Read
180
181     .globl Write
182     .ent    Write
183 Write:
184     addiu $2, $0, SC_Write
185     syscall
186     j   $31
187     .end Write
188
189     .globl Close
190     .ent    Close
191 Close:
192     addiu $2, $0, SC_Close
193     syscall
194     j   $31
195     .end Close
```

參照上方其他函式的MIPS呼叫方式，增加Open、Read、Write、Close四個區域作為呼叫入口，運作方式與 (a)部分start.S的SC_Halt程式碼相同。

(3). userprog/exception.cc

在這個檔案裡，主要修改的部分為ExceptionHandler()這個函數。

```
50  void
51  ExceptionHandler(ExceptionType which)
52  {
53      char ch;
54      int val;
55      int type = kernel->machine->ReadRegister(2);
56      int status, exit, threadID, programID, fileID, numChar;
57      DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
58      DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which <<
59      switch (which) {
60      case SyscallException:
61      switch(type) {
62          case SC_Halt:
63          DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
64          SysHalt();
65          cout<<"in exception\n";
66          ASSERTNOTREACHED();
67          break;
```

ExceptionHandler()相關介紹已於上方part (b)中完成，在這裡我們實作的部分是增加4個SyscallException的case（SC_Open、SC_Write、SC_Read、SC_Close）。

以下依序介紹：

1、4. SC_Open（SC_Close高度相似）

```
108         case SC_Open:
109             val = kernel->machine->ReadRegister(4);
110             {
111                 char *name = &(kernel->machine->mainMemory[val]);
112                 status = SysOpen(name);
113                 kernel->machine->WriteRegister(2, (int)status);
114             }
115             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
116             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
117             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
118             return;
119             ASSERTNOTREACHED();
120             break;
```

system call的相關資訊存在2、4號register裡面。

open檔案需要以檔名為參數，讀取方式參造上方SC_create的case，利用4號register為key查找main memory的value存放給name變數。

[112]呼叫ksyscall.h中的SysOpen()函數，處理SC_Open。

SC_close實作方式同理。

2、3. SC_Write（SC_Read高度相似）

```
124         case SC_Write:
125             val = kernel->machine->ReadRegister(4);
126             {
127                 char *buffer = &(kernel->machine->mainMemory[val]);
128                 status = SysWrite(buffer, kernel->machine->ReadRegister(5), kernel->machine->ReadRegister(6));
129                 kernel->machine->WriteRegister(2, (int)status);
130             }
131             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
132             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
133             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
134             return;
135             ASSERTNOTREACHED();
136             break;
```

system call的相關資訊被存在2、4、5、6、7號register裡面。

2號register存放system call type；其他register存的內容則不一定，以每個函數的的要求為準，如Write就分別以4、5、6號分別存放char *buffer、int size、OpenFileId id。

[128]呼叫ksyscall.h中的SysWrite()函數，處理SC_Write。

(4). userprog/ksyscall.h

```
60 OpenFileId SysOpen(char *name)
61 {
62   return kernel->fileSystem->OpenAFile(name);
63 }
```

```
44 int SysRead(char *buffer, int size, OpenFileId id)
45 {
46   return kernel->fileSystem->ReadFile(buffer, size, id);
47 }
48 int SysClose(OpenFileId id)
49 {
50   return kernel->fileSystem->CloseFile(id);
51 }
52 int SysWrite(char *buffer, int size, OpenFileId id)
53 {
54   return kernel->fileSystem->WriteFile(buffer, size, id);
55 }
```

ksyscall.h呼叫filesys/filesys.h中的OpenAFile()、ReadFile()、CloseFile()、WriteFile()等函數進行實作細節。

(5). filesys/filesys.h

此檔案為最主要的實作，實作函數為

OpenFileId OpenAFile(char *name) 、

int WriteFile(char *buffer, int size, OpenFileId id) 、

int ReadFile(char *buffer, int size, OpenFileId id) 、

int CloseFile(OpenFileId id)。

1. OpenFileId OpenAFile(char *name)

```
76      OpenFileId OpenAFile(char *name)
77      { //周昱宏改的 這四個原本有註解
78          int i;
79          for (i = 0; i < 20; i++)
80          {
81              if (!OpenFileTable[i])
82              {
83                  if (OpenFileTable[i] = Open(name))
84                  {
85                      return i;
86                  }
87                  else
88                  {
89                      return -1;
90                  }
91              }
92          }
93          if (i == 20)
94              return -1;
95      }
```

OpenAFile函式的目標是要利用檔名變數開啟檔案，並維護OpenFileTable，最後回傳屬於該檔案的OpenFieldId。因此我的實作方式是遍歷全部的OpenFileTable，若其中任一元素為NULL則利用上方已定義好的Open函式將檔案打開，若Open函式回傳OpenFile *型態則表示開啟成功，將其指定給該元素，並回傳該元素id，若回傳NULL則表示開啟失敗，回傳-1。若OpenFileTable所有元素皆不是NULL，代表Table滿了即回傳-1。

2. WriteFile(char *buffer, int size, OpenFileId id)

3. ReadFile(char *buffer, int size, OpenFileId id)

```
96      int WriteFile(char *buffer, int size, OpenFileId id)
97      {
98          OpenFile *openfile = OpenFileTable[id];
99          if (openfile == NULL)
100             return -1;
101         int NumWrite = openfile->Write(buffer, size);
102         return NumWrite;
103     }
104     int ReadFile(char *buffer, int size, OpenFileId id) /
105     {
106         OpenFile *openfile = OpenFileTable[id];
107         if (openfile == NULL)
108             return -1;
109         int NumRead = openfile->Read(buffer, size);
110         return NumRead;
111     }
```

[98、106] 透過OpenFileTable以及id取得想要write或read的檔案。

[99、107] 如果失敗回傳-1。

[101、109] 根據檔案上方註解// Operations on an individual "open" file (read, write, close) are to be found in the OpenFile class (openfile.h)，這裡其實是呼叫了被定義於openfile.h中的函數Write()和Read()來幫助我們完成計算寫或讀了幾個characters的工作。

[102、110] 回傳寫或讀了幾個characters。

4. int CloseFile(OpenFileId id)

```
112     int CloseFile(OpenFileId id) // 李彥璋改的
113     {
114         OpenFile *openfile = OpenFileTable[id];
115         if (openfile == NULL)
116             return -1;
117         OpenFileTable[id] = NULL;
118         delete openfile;
119         return 1;
120     }
```

[114] 透過OpenFileTable以及id取得想要close的檔案。

[115] 如果失敗回傳-1。

[116] 將OpenFileTable該檔案的位子改成NULL。

[117] delete the OpenFile after closeing the file。

[118] 如果成功回傳1。

測試成功

```
Last login: Fri Oct 22 19:11:51 2021 from 10.121.186.106
[os21team56@localhost ~]$ cd NachOS-4.0_MP1/code/test
[os21team56@localhost test]$ ../build.linux/nachos -e fileIO_test1
fileIO_test1
Success on creating file1.test
Machine halting!

This is halt
Ticks: total 954, idle 0, system 130, user 824
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
[os21team56@localhost test]$ ../build.linux/nachos -e fileIO_test2
fileIO_test2
Passed! ^_^
Machine halting!

This is halt
Ticks: total 815, idle 0, system 120, user 695
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
[os21team56@localhost test]$
```

(d). What difficulties did you encounter, when implementing this assignment?

1. 有許多函式不清楚定義與實作的檔案位置，或是有重複多個的宣告，必須查閱很多資料才能了解。
2. 一開始不太清楚MIPS指令集架構。
3. 不清楚trace code需要詳細到什麼程度。
4. 和組員的程式整合。

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