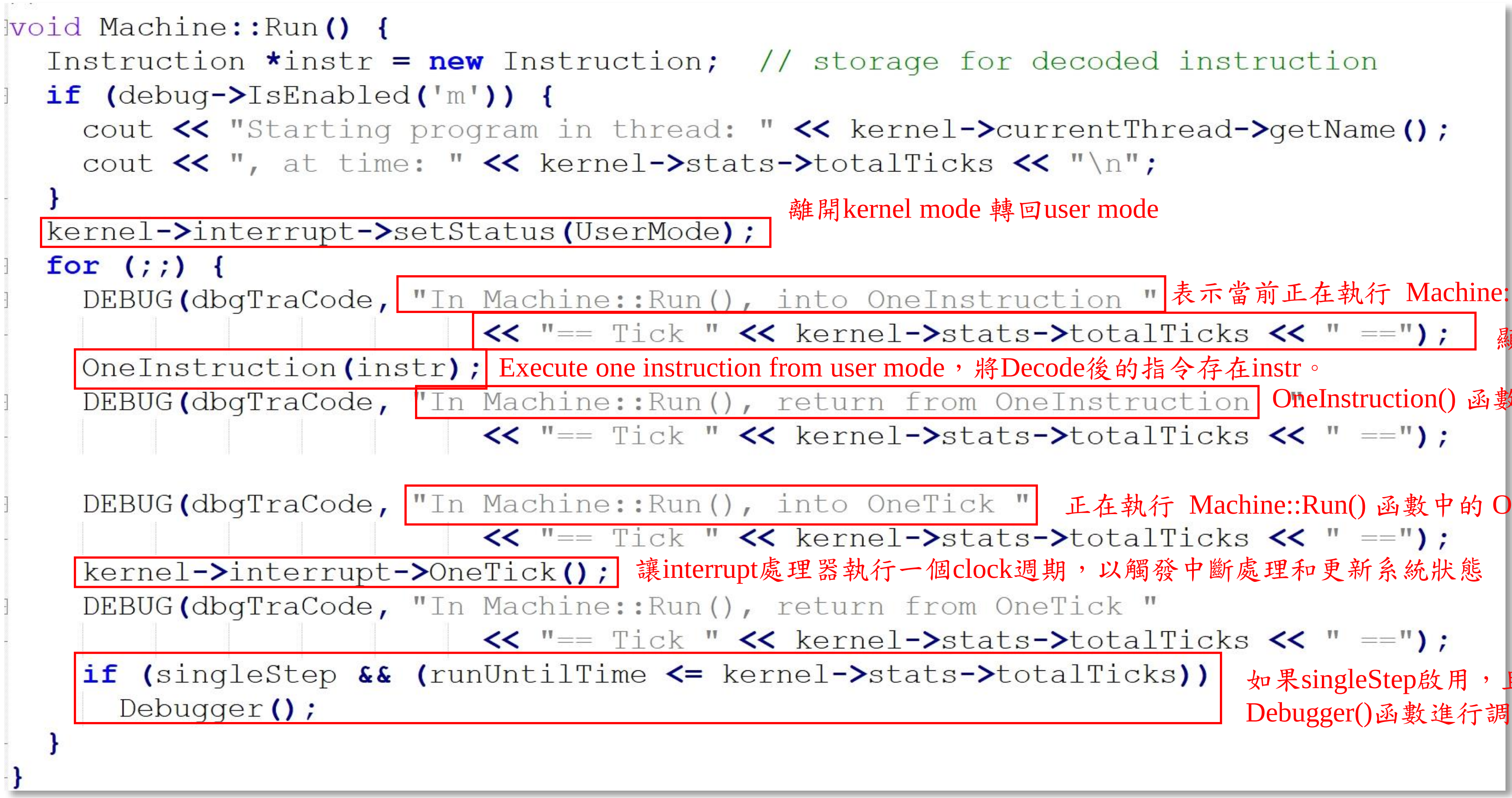
MP1 System call

111062502

凃博允

SC_Halt Trace code: machine/mipssim.cc



```
void Machine::Run() {
  Instruction *instr = new Instruction;  // storage for decoded 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();
  }
}
```

離開kernel mode 轉回user mode

表示當前正在執行 Machine::Run() 函數中的 OneInstruction() 子函數

顯示NachOS Kernel運行時間

Execute one instruction from user mode，將Decode後的指令存在instr。

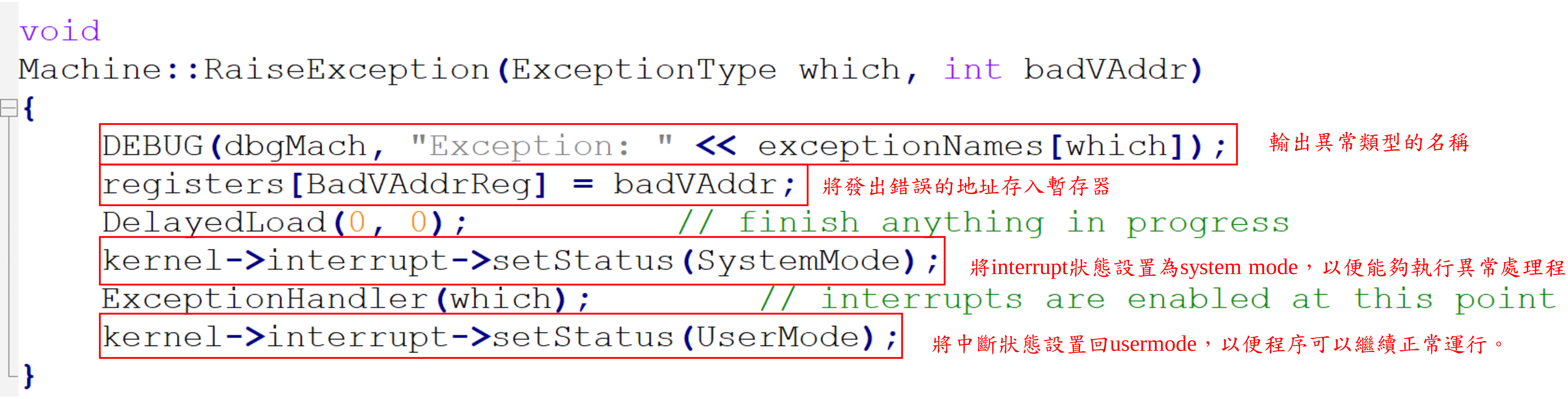
OneInstruction() 函數已經執行完畢，返回到 Machine::Run() 函數

正在執行 Machine::Run() 函數中的 OneTick() 子函數

讓interrupt處理器執行一個clock週期，以觸發中斷處理和更新系統狀態

如果singleStep啟用，且小於等於當前系統運行時間，則調用 Debugger()函數進行調試

SC_Halt Trace code: machine/machine.cc



```
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);
    ExceptionHandler(which);             // interrupts are enabled at this point
    kernel->interrupt->setStatus(UserMode);
}
```

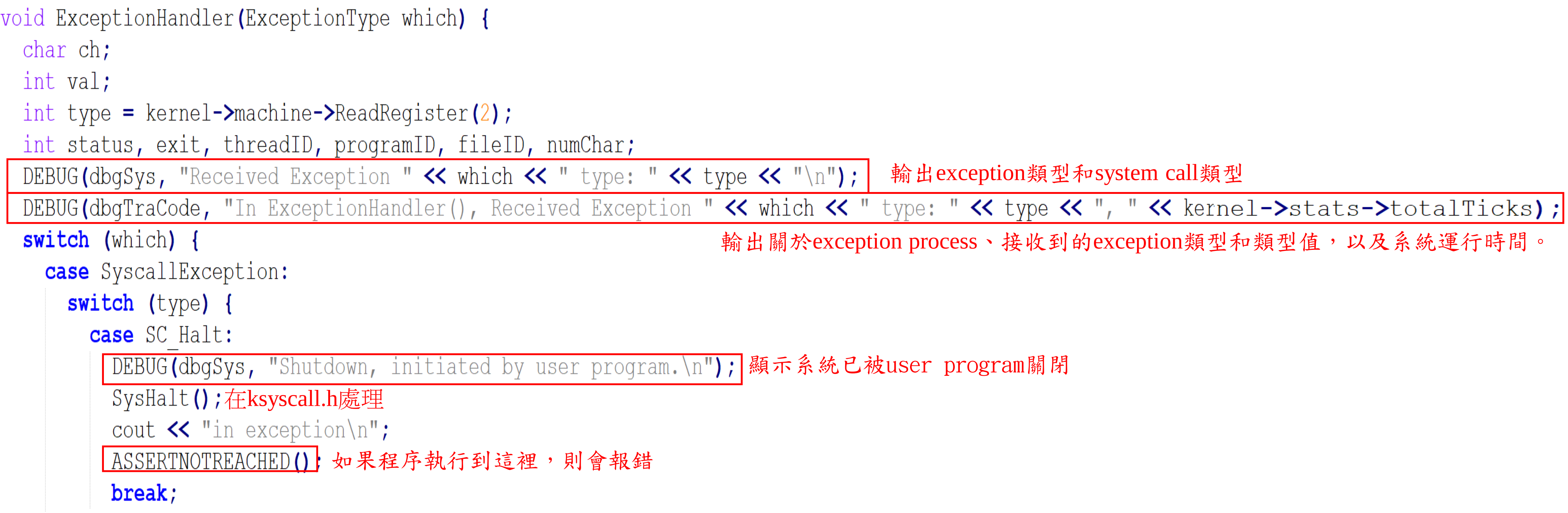
輸出異常類型的名稱

將發出錯誤的地址存入暫存器

將interrupt狀態設置為system mode，以便能夠執行異常處理程序。

將中斷狀態設置回usermode，以便程序可以繼續正常運行。

SC_Halt Trace code: userprog/exception.cc



```
void ExceptionHandler(ExceptionType which) {
  char ch;
  int val;
  int type = kernel->machine->ReadRegister(2);
  int status, exit, threadID, programID, fileID, numChar;
  DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
  DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which << " type: " << type << ", " << kernel->stats->totalTicks);
  switch (which) {
    case SyscallException:
      switch (type) {
        case SC_Halt:
          DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
          SysHalt();
          cout << "in exception\n";
          ASSERTNOTREACHED();
          break;
```

輸出exception類型和system call類型

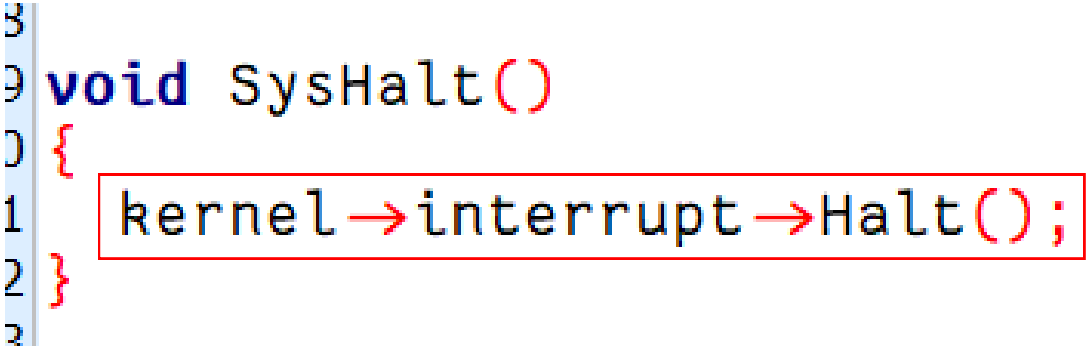
輸出關於exception process、接收到的exception類型和類型值，以及系統運行時間。

顯示系統已被user program關閉

在ksyscall.h處理

如果程序執行到這裡，則會報錯

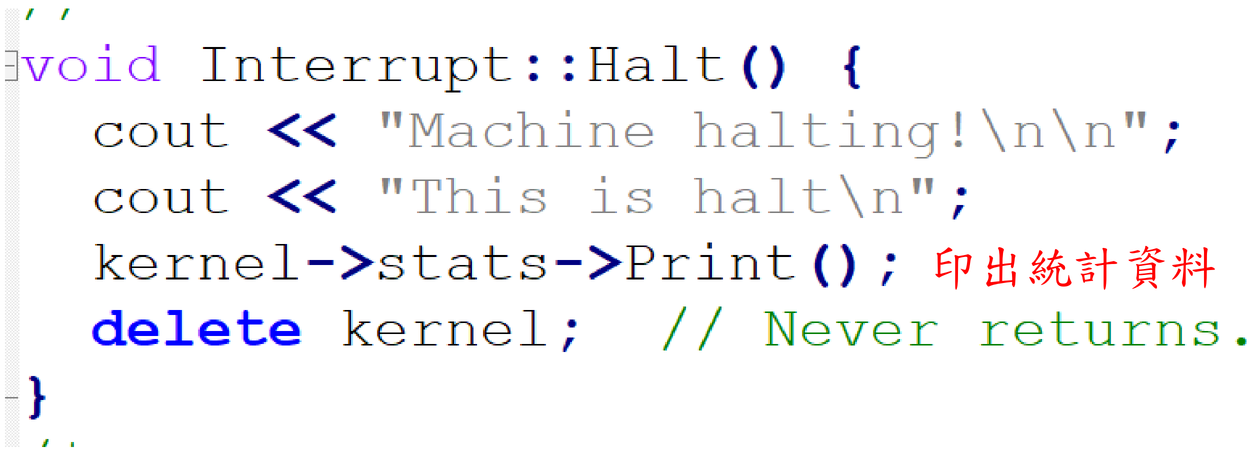
SC_Halt Trace code: userprog/ksycall.h



```
void SysHalt()
{
    kernel→interrupt→Halt();
}
```

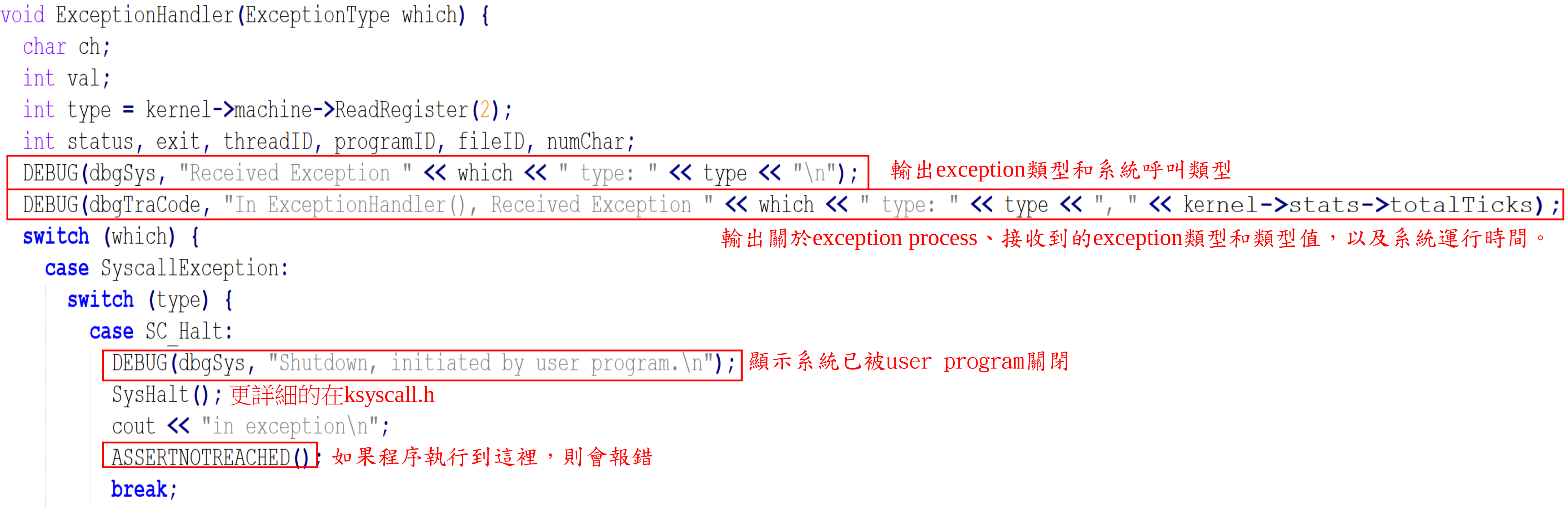
停止 NachOS 系統

SC_Halt Trace code: machine/interrupt.cc



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

SC_Create Trace code: userprog/exception.cc



```
void ExceptionHandler(ExceptionType which) {
  char ch;
  int val;
  int type = kernel->machine->ReadRegister(2);
  int status, exit, threadID, programID, fileID, numChar;
  DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
  DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which << " type: " << type << ", " << kernel->stats->totalTicks);
  switch (which) {
    case SyscallException:
      switch (type) {
        case SC_Halt:
          DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
          SysHalt();
          cout << "in exception\n";
          ASSERTNOTREACHED();
          break;
```

輸出exception類型和系統呼叫類型

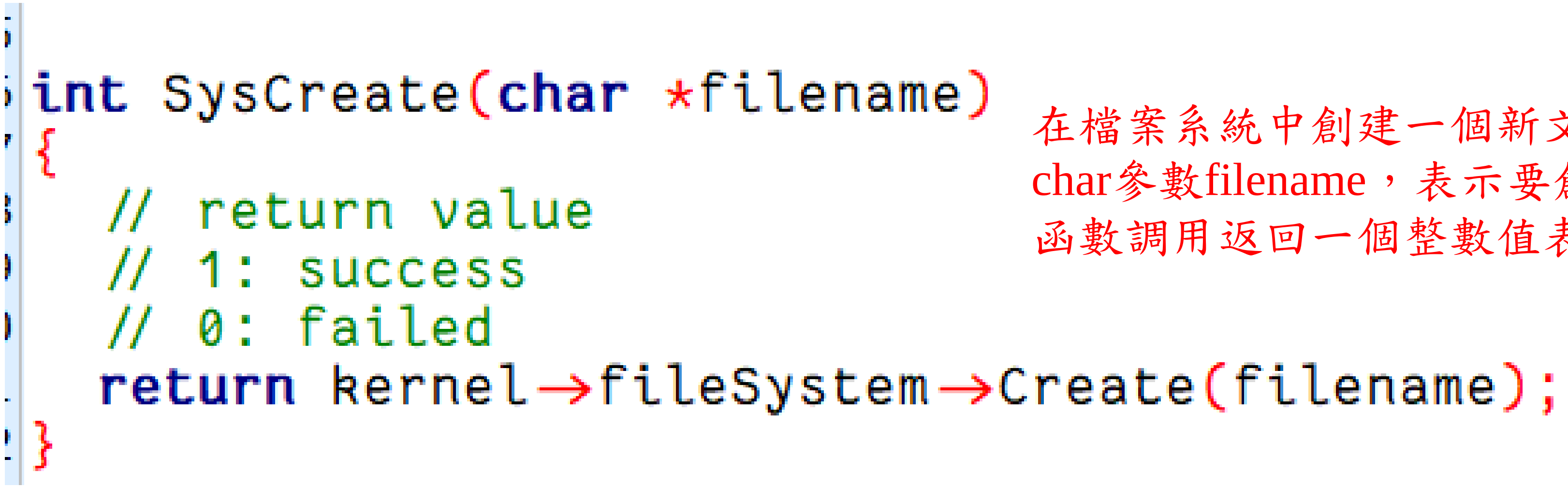
輸出關於exception process、接收到的exception類型和類型值，以及系統運行時間。

顯示系統已被user program關閉

更詳細的在ksyscall.h

如果程序執行到這裡，則會報錯

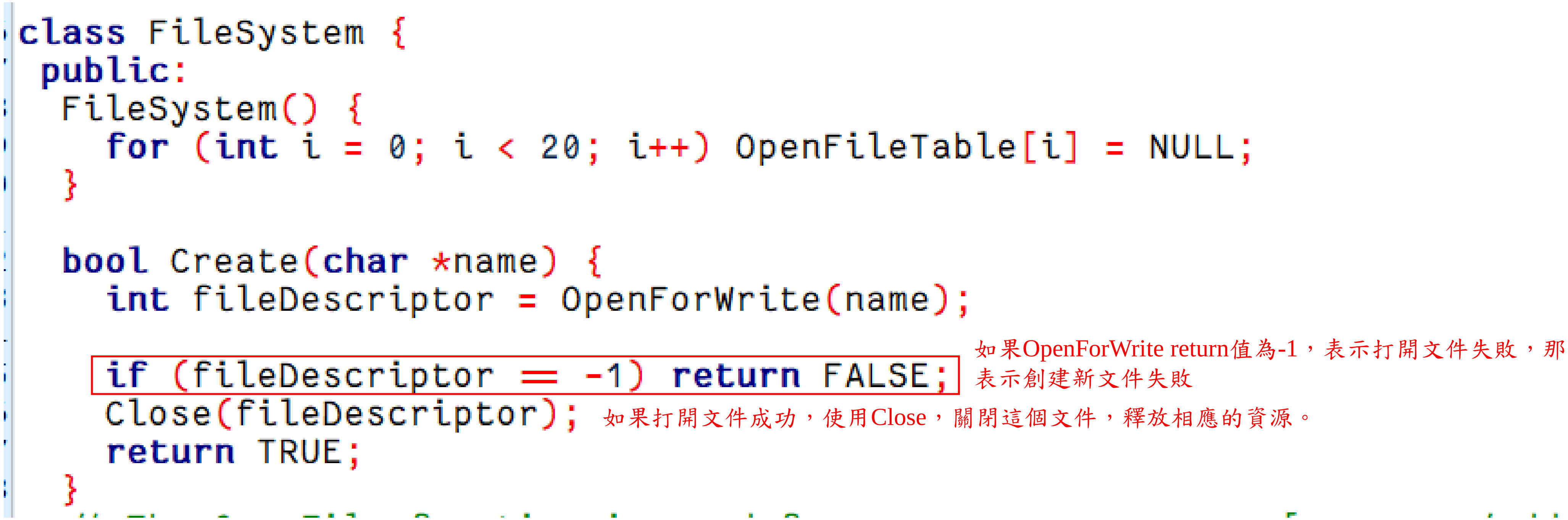
SC_Create Trace code: userprog/ksyscall.h



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

在檔案系統中創建一個新文件。該函數接受char參數filename，表示要創建的新文件的名稱，函數調用返回一個整數值表示創建操作的結果

SC_Create Trace code: filesys/filesys.h

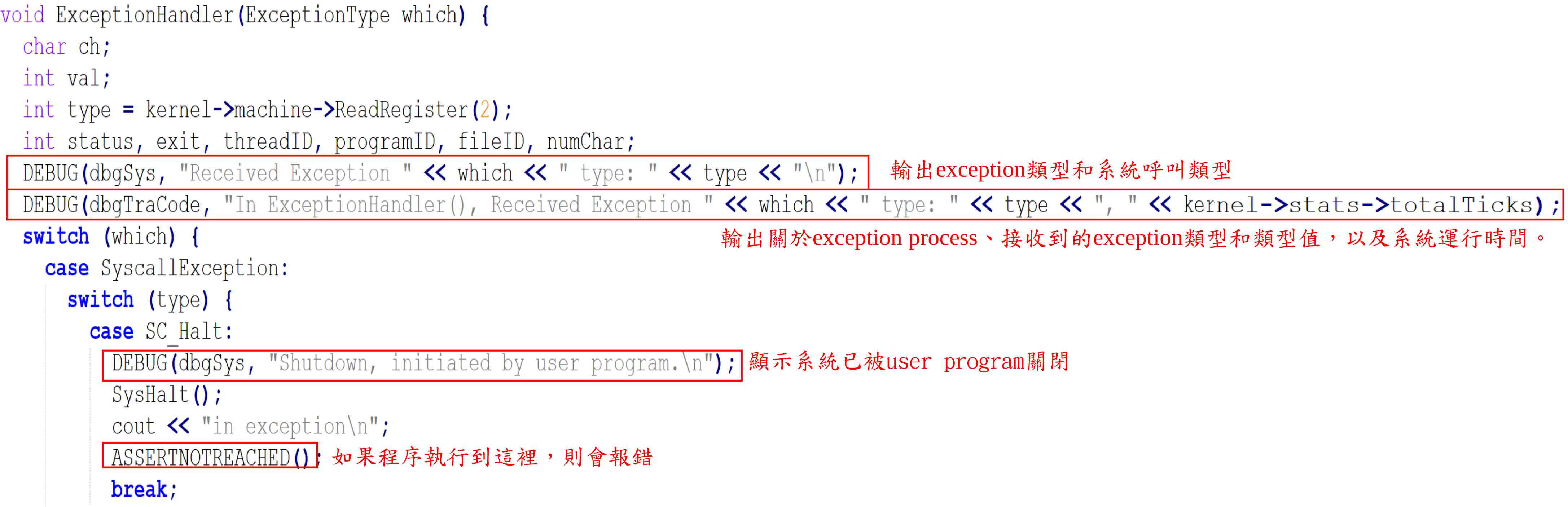


```
class FileSystem {
 public:
  FileSystem() {
    for (int i = 0; i < 20; i++) OpenFileTable[i] = NULL;
  }

  bool Create(char *name) {
    int fileDescriptor = OpenForWrite(name);

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

如果OpenForWrite return值為-1，表示打開文件失敗，那麼直接return FALSE，表示創建新文件失敗

如果打開文件成功，使用Close，關閉這個文件，釋放相應的資源。

SC_PrintInt Trace code: userprog/exception.cc



```
void ExceptionHandler(ExceptionType which) {
  char ch;
  int val;
  int type = kernel->machine->ReadRegister(2);
  int status, exit, threadID, programID, fileID, numChar;
  DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");
  DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which << " type: " << type << ", " << kernel->stats->totalTicks);
  switch (which) {
    case SyscallException:
      switch (type) {
        case SC_Halt:
          DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
          SysHalt();
          cout << "in exception\n";
          ASSERTNOTREACHED();
          break;
```

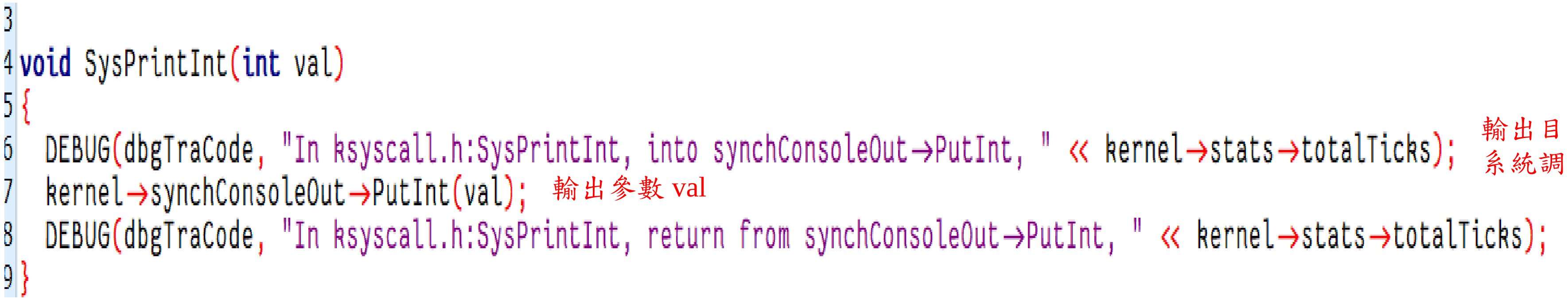
輸出exception類型和系統呼叫類型

輸出關於exception process、接收到的exception類型和類型值，以及系統運行時間。

顯示系統已被user program關閉

如果程序執行到這裡，則會報錯

SC_PrintInt Trace code: userprog/ksyscall.h



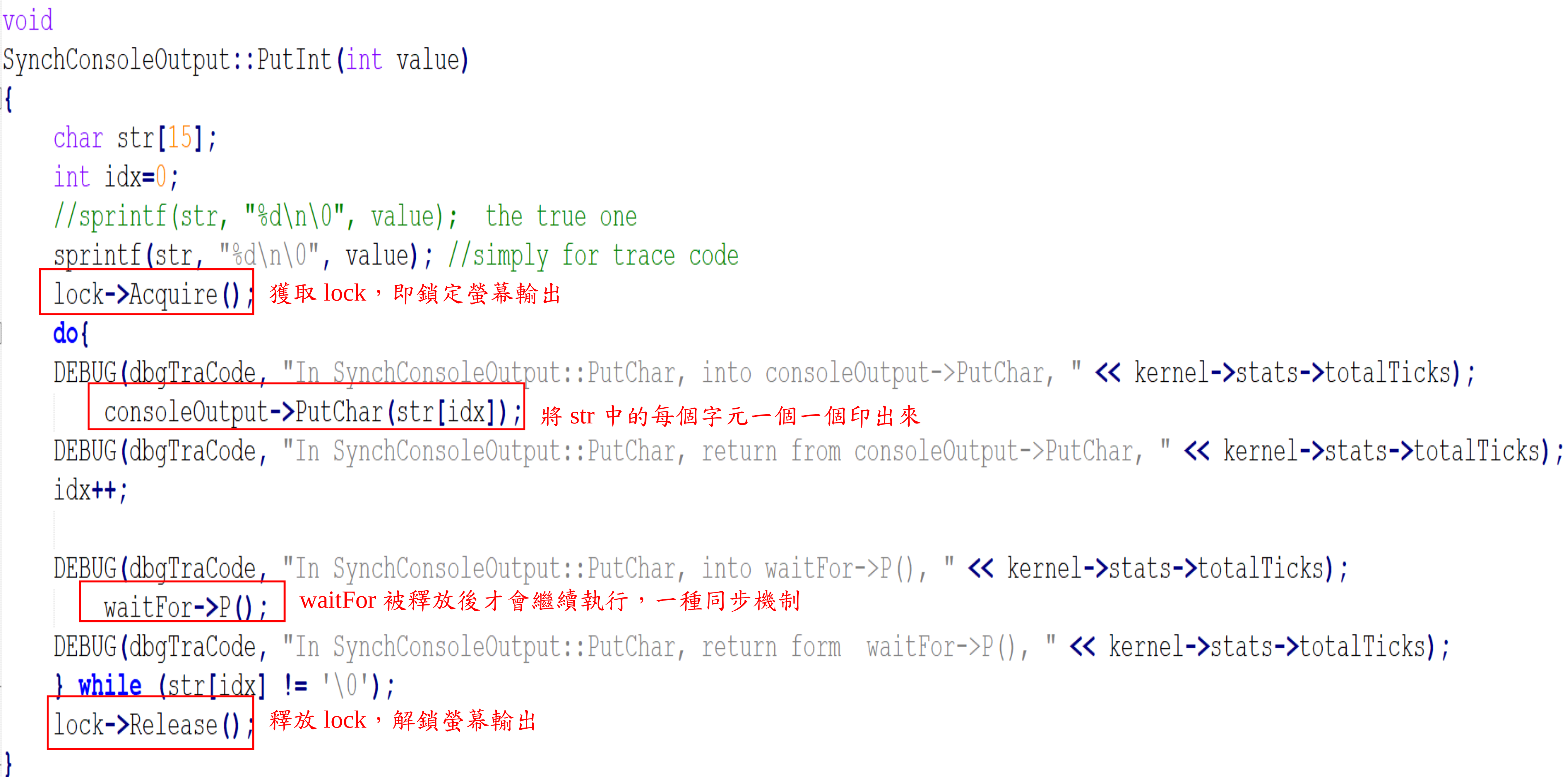
```

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);
}
```

輸出目前系統運行的總時脈數，以及目前正在執行的系統調用的相關訊息。

輸出參數 val

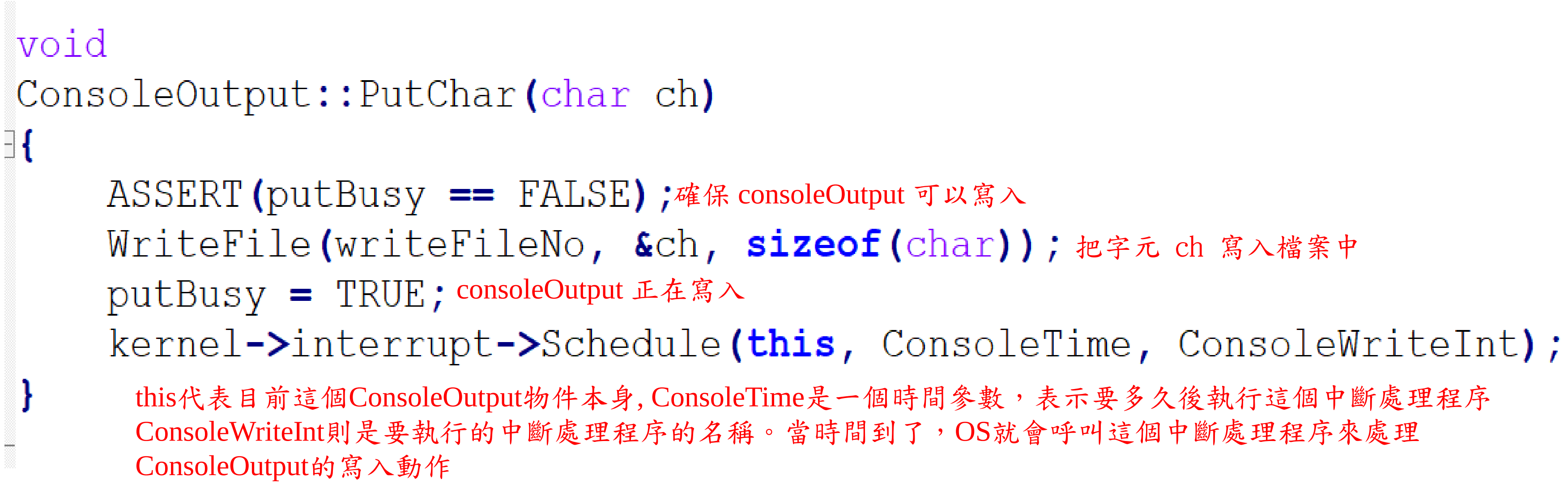
輸出表示從 synchConsoleOut->PutInt 中返回了，以及目前系統運行的總時脈數

SC_PrintInt Trace code: userprog/synchconsole.cc



```
void
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(); 獲取 lock，即鎖定螢幕輸出
    do{
    DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into consoleOutput->PutChar, " << kernel->stats->totalTicks);
        consoleOutput->PutChar(str[idx]); 將 str 中的每個字元一個一個印出來
    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(); waitFor 被釋放後才會繼續執行，一種同步機制
    DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return form  waitFor->P(), " << kernel->stats->totalTicks);
    } while (str[idx] != '\0');
    lock->Release(); 釋放 lock，解鎖螢幕輸出
}
```

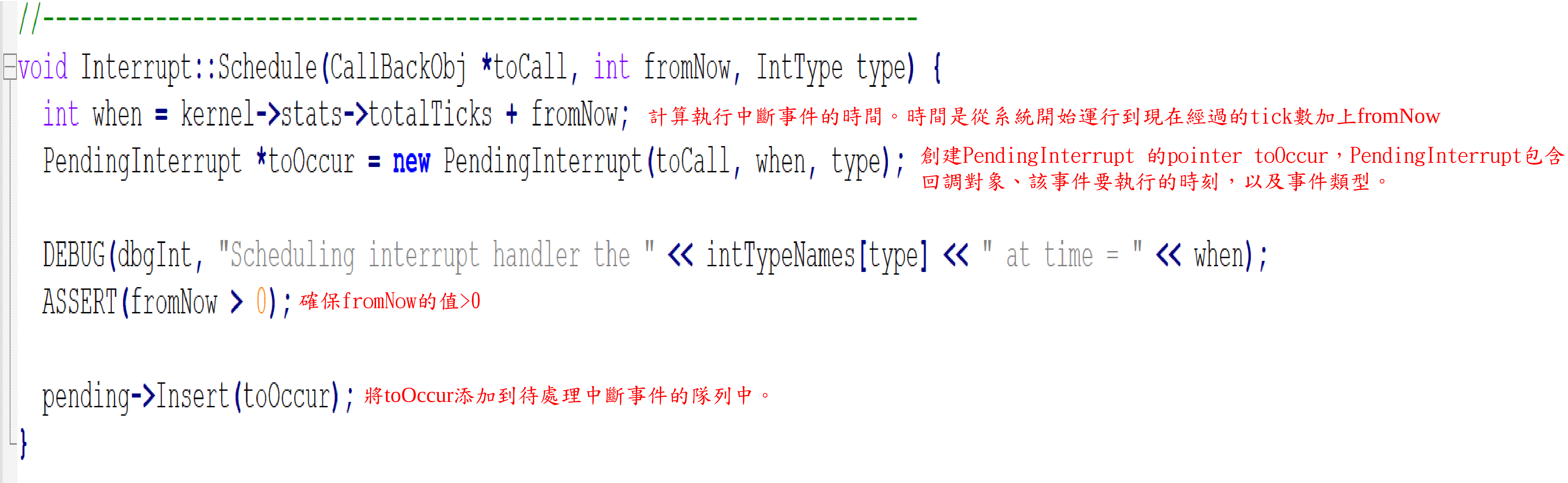
SC_PrintInt Trace code: machine/console.cc



```
void
ConsoleOutput::PutChar(char ch)
{
    ASSERT(putBusy == FALSE);確保 consoleOutput 可以寫入
    WriteFile(writeFileNo, &ch, sizeof(char)); 把字元 ch 寫入檔案中
    putBusy = TRUE; consoleOutput 正在寫入
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}
```

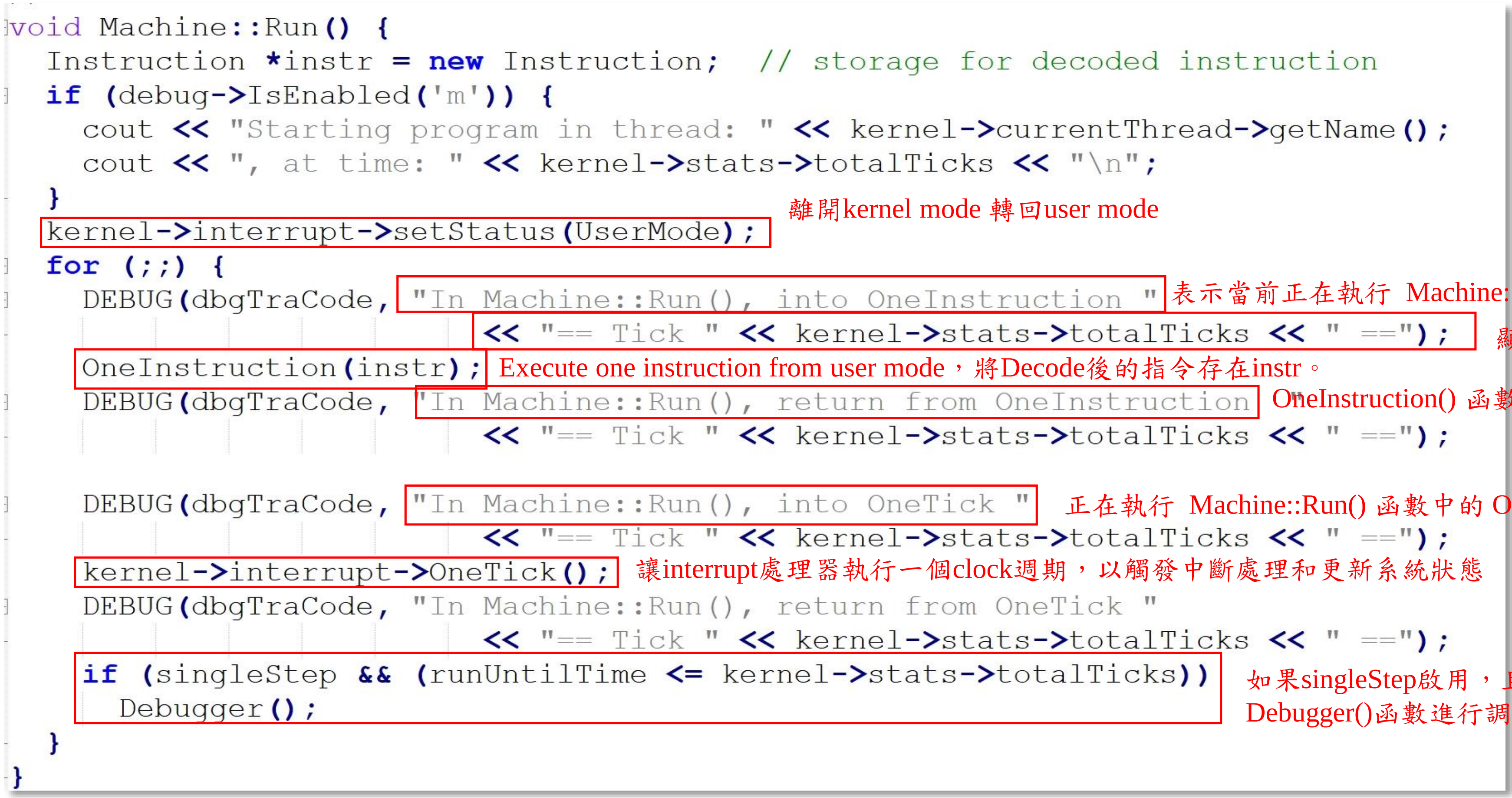
this代表目前這個ConsoleOutput物件本身, ConsoleTime是一個時間參數，表示要多久後執行這個中斷處理程序
ConsoleWriteInt則是要執行的中斷處理程序的名稱。當時間到了，OS就會呼叫這個中斷處理程序來處理
ConsoleOutput的寫入動作

SC_PrintInt Trace code: machine/interrupt.cc



```
//----------------------------------------------------------------------
void Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type) {
  int when = kernel->stats->totalTicks + fromNow; 計算執行中斷事件的時間。時間是從系統開始運行到現在經過的tick數加上fromNow
  PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type); 創建PendingInterrupt 的pointer toOccur，PendingInterrupt包含
                                                                        回調對象、該事件要執行的時刻，以及事件類型。

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


  pending->Insert(toOccur); 將toOccur添加到待處理中斷事件的隊列中。
}
```

SC_PrintInt Trace code: machine/mipssim.cc



```
void Machine::Run() {
  Instruction *instr = new Instruction;  // storage for decoded 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();
  }
}
```

離開kernel mode 轉回user mode

表示當前正在執行 Machine::Run() 函數中的 OneInstruction() 子函數

顯示NachOS Kernel運行時間

Execute one instruction from user mode，將Decode後的指令存在instr。

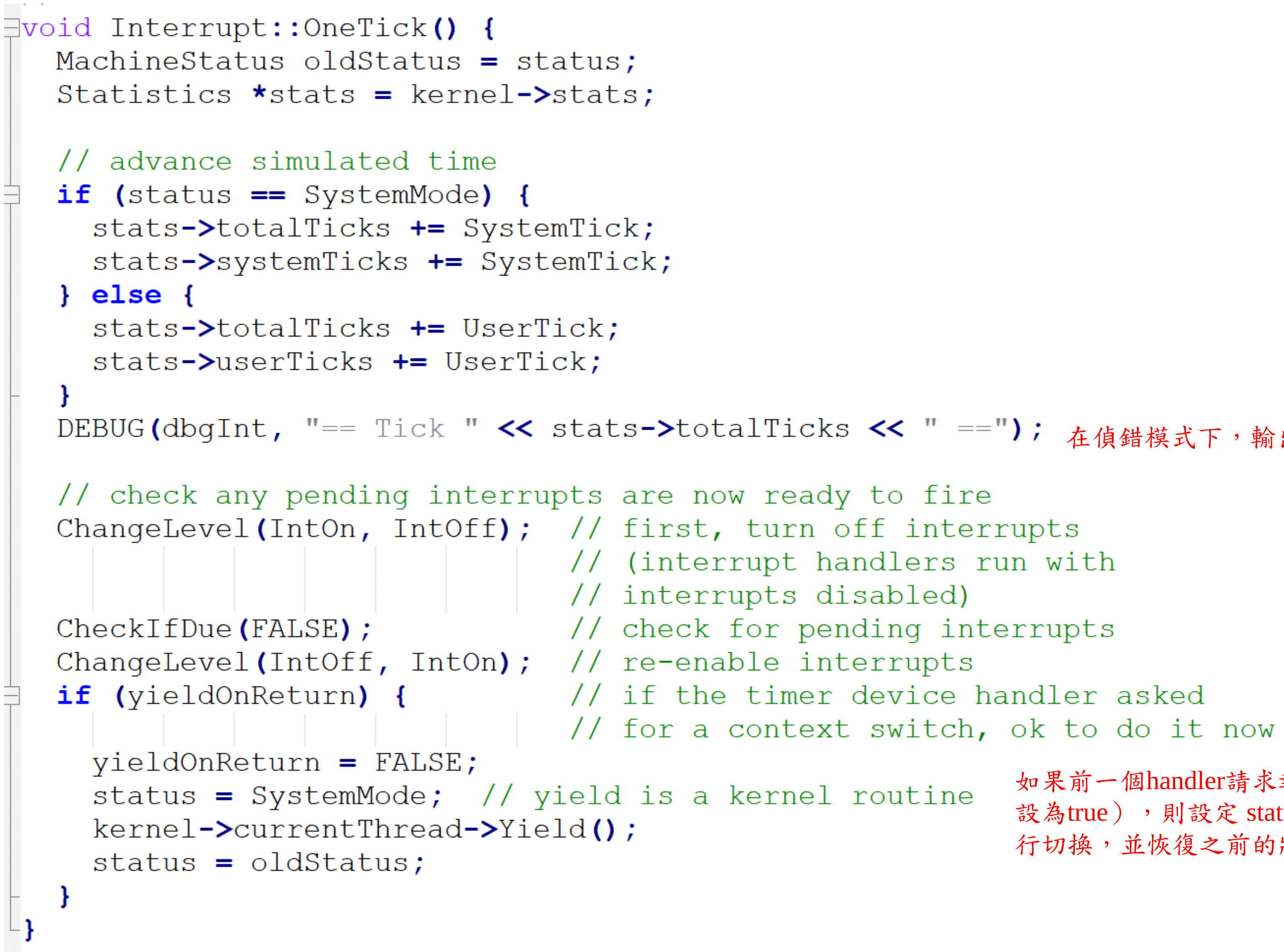
OneInstruction() 函數已經執行完畢，返回到 Machine::Run() 函數

正在執行 Machine::Run() 函數中的 OneTick() 子函數

讓interrupt處理器執行一個clock週期，以觸發中斷處理和更新系統狀態

如果singleStep啟用，且小於等於當前系統運行時間，則調用Debugger()函數進行調試

SC_PrintInt Trace code: machine/interrupt.cc

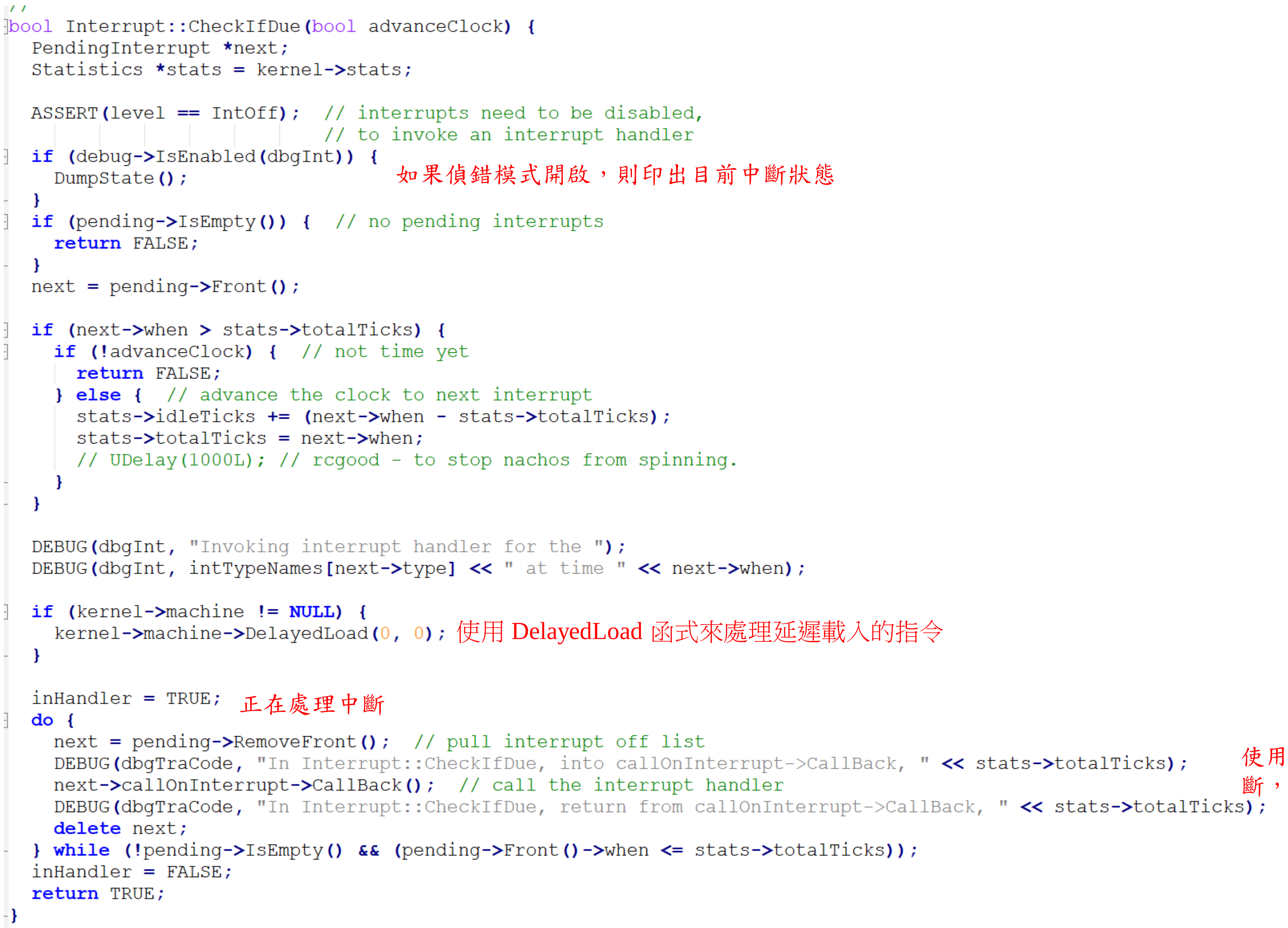


```
void Interrupt::OneTick() {
  MachineStatus oldStatus = status;
  Statistics *stats = kernel->stats;

  // advance simulated time
  if (status == SystemMode) {
    stats->totalTicks += SystemTick;
    stats->systemTicks += SystemTick;
  } else {
    stats->totalTicks += UserTick;
    stats->userTicks += UserTick;
  }
  DEBUG(dbgInt, "== Tick " << stats->totalTicks << " ==");

  // check any pending interrupts are now ready to fire
  ChangeLevel(IntOn, IntOff);  // first, turn off interrupts
                               // (interrupt handlers run with
                               // interrupts disabled)
  CheckIfDue(FALSE);           // check for pending interrupts
  ChangeLevel(IntOff, IntOn);  // re-enable interrupts
  if (yieldOnReturn) {         // if the timer device handler asked
                               // for a context switch, ok to do it now
    yieldOnReturn = FALSE;
    status = SystemMode;  // yield is a kernel routine
    kernel->currentThread->Yield();
    status = oldStatus;
  }
}
```

在偵錯模式下，輸出目前的總時脈

如果前一個handler請求執行緒切換（yieldOnReturn 被設為true），則設定 status 為SystemMode，執行緒進行切換，並恢復之前的狀態

SC_PrintInt Trace code: machine/interrupt.cc



```
//
bool Interrupt::CheckIfDue(bool advanceClock) {
    PendingInterrupt *next;
    Statistics *stats = kernel->stats;

    ASSERT(level == IntOff);  // interrupts need to be disabled,
                              // to invoke an interrupt handler
    if (debug->IsEnabled(dbgInt)) {
      DumpState();
    }
    if (pending->IsEmpty()) {   // no pending interrupts
      return FALSE;
    }
    next = pending->Front();

    if (next->when > stats->totalTicks) {
      if (!advanceClock) {   // not time yet
        return FALSE;
      } else {   // advance the clock to next interrupt
        stats->idleTicks += (next->when - stats->totalTicks);
        stats->totalTicks = next->when;
        // UDelay(1000L); // rcgood - to stop nachos from spinning.
      }
    }

    DEBUG(dbgInt, "Invoking interrupt handler for the ");
    DEBUG(dbgInt, intTypeNames[next->type] << " at time " << next->when);

    if (kernel->machine != NULL) {
      kernel->machine->DelayedLoad(0, 0);
    }

    inHandler = TRUE;
    do {
      next = pending->RemoveFront();  // pull interrupt off list
      DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, into callOnInterrupt->CallBack, " << stats->totalTicks);
      next->callOnInterrupt->CallBack();  // call the interrupt handler
      DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, return from callOnInterrupt->CallBack, " << stats->totalTicks);
      delete next;
    } while (!pending->IsEmpty() && (pending->Front()->when <= stats->totalTicks));
    inHandler = FALSE;
    return TRUE;
}
```

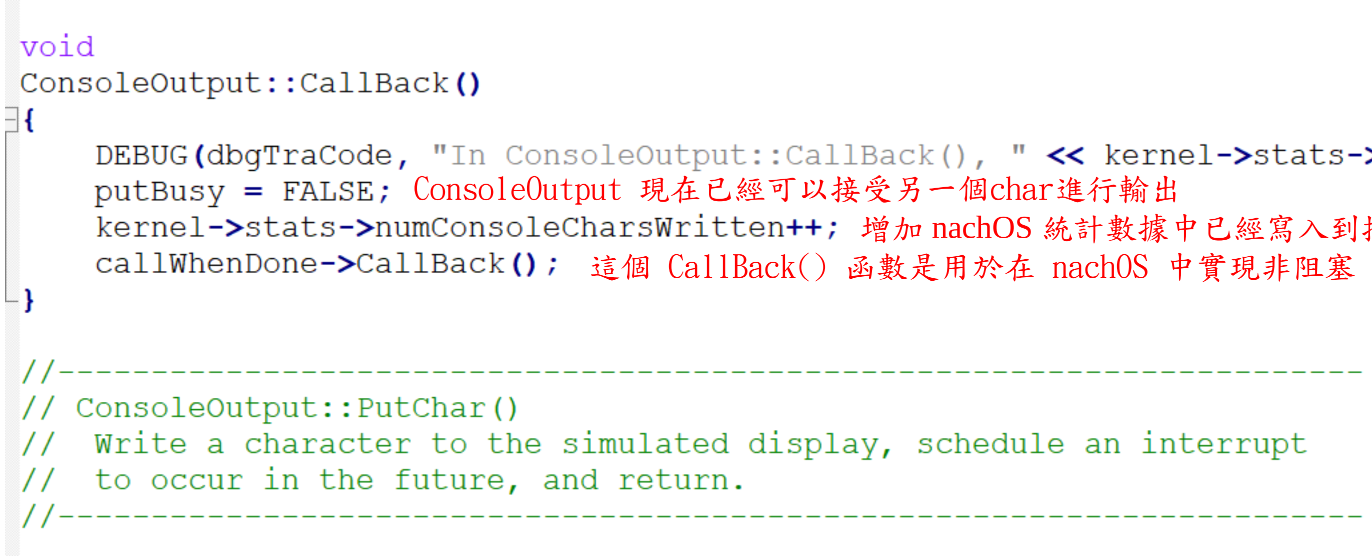
如果偵錯模式開啟，則印出目前中斷狀態

使用 DelayedLoad 函式來處理延遲載入的指令

正在處理中斷

使用 while 迴圈從等待中斷佇列中移除所有已經到達時間點的中斷，並逐一呼叫CallBack 函式，最後釋放 PendingInterrupt 物件

SC_PrintInt Trace code: machine/console.cc



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

//----------------------------------------------------------------------
// ConsoleOutput::PutChar()
//  Write a character to the simulated display, schedule an interrupt
//  to occur in the future, and return.
//----------------------------------------------------------------------
```

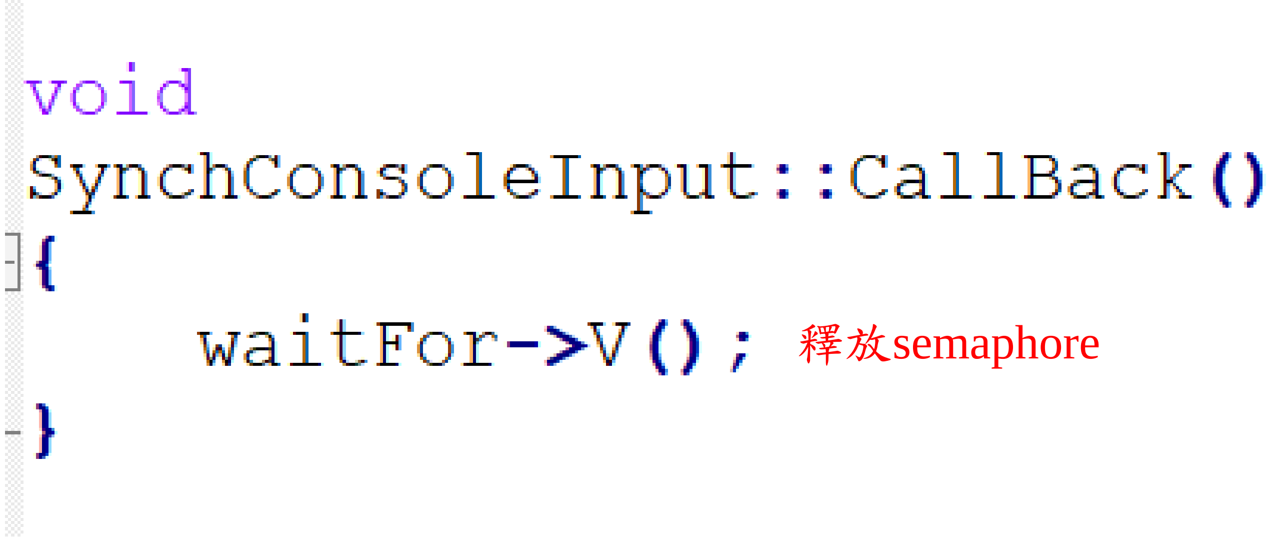
已經進入 ConsoleOutput 的 CallBack() 函數，
並顯示當前的 totalTicks。

ConsoleOutput 現在已經可以接受另一個char進行輸出

增加 nachOS 統計數據中已經寫入到控制台的char數量

這個 CallBack() 函數是用於在 nachOS 中實現非阻塞 IO 的機制

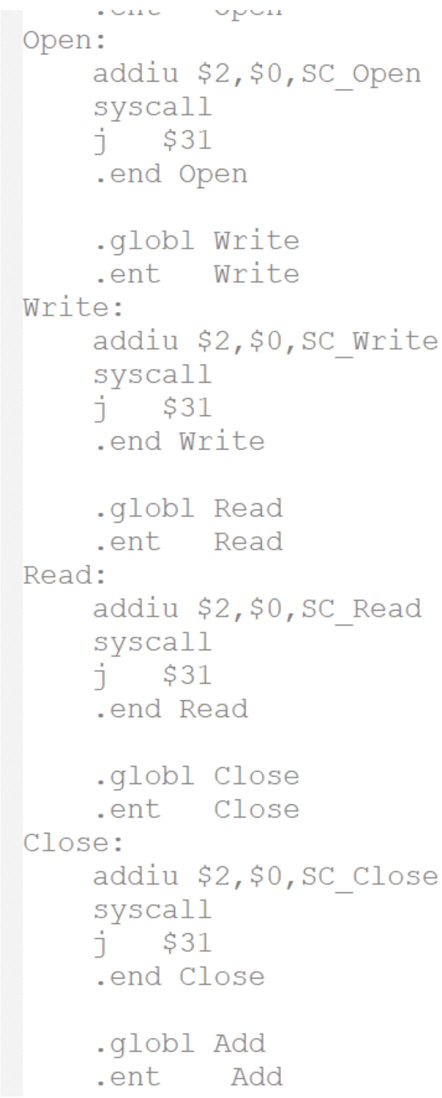
SC_PrintInt Trace code: userprog/synchconsole.cc



```
void
SynchConsoleInput::CallBack()
{
    waitFor->V();
}
```

釋放semaphore

SC_fileIO_test1 Trace code: test/start.S



```
    .ent    Open
Open:
    addiu $2,$0,SC_Open
    syscall
    j   $31
    .end Open

    .globl Write
    .ent   Write
Write:
    addiu $2,$0,SC_Write
    syscall
    j   $31
    .end Write

    .globl Read
    .ent   Read
Read:
    addiu $2,$0,SC_Read
    syscall
    j   $31
    .end Read

    .globl Close
    .ent   Close
Close:
    addiu $2,$0,SC_Close
    syscall
    j   $31
    .end Close

    .globl Add
    .ent    Add
```

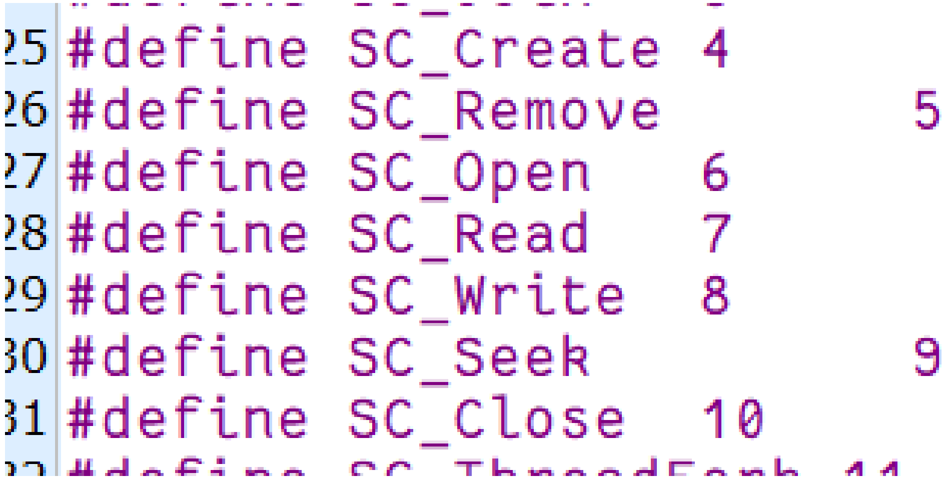
$v0 = 0 + SC_Open

進到 syscall handler 處理 print

jump to return register, 即 “return”，執行system call

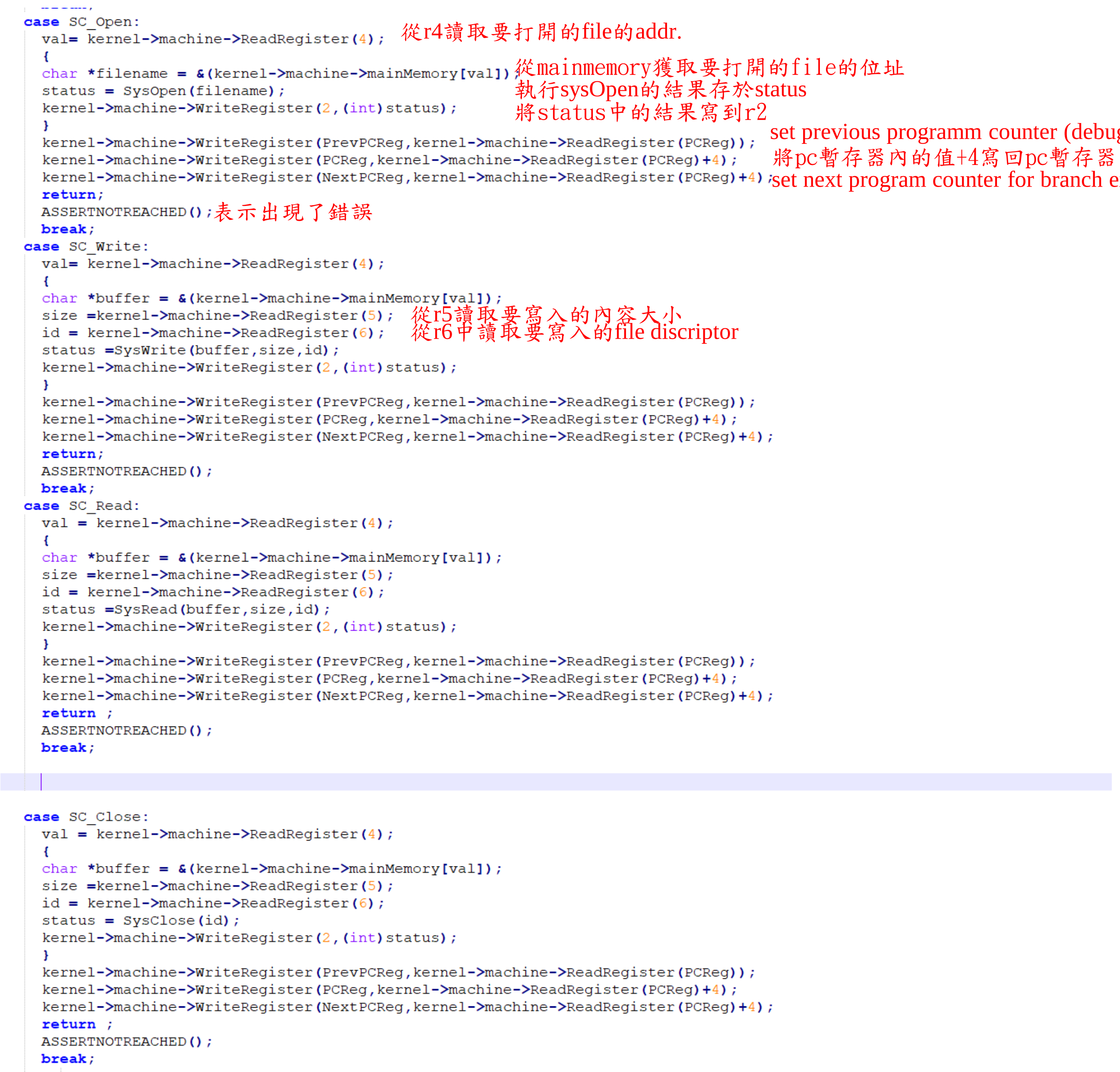
SC_fileIO_test1 Trace code: userprog/syscall.h

#define system call進入exception所表示的編碼



```
#define SC_Create 4
#define SC_Remove       5
#define SC_Open   6
#define SC_Read   7
#define SC_Write  8
#define SC_Seek         9
#define SC_Close  10
#define SC_ThreadFork 11
```

SC_fileIO_test1 Trace code: userprog/exception.cc



```
case SC_Open:
  val= kernel->machine->ReadRegister(4);
  {
  char *filename = &(kernel->machine->mainMemory[val]);
  status = SysOpen(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;
case SC_Write:
  val= kernel->machine->ReadRegister(4);
  {
  char *buffer = &(kernel->machine->mainMemory[val]);
  size =kernel->machine->ReadRegister(5);
  id = kernel->machine->ReadRegister(6);
  status =SysWrite(buffer,size,id);
  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;
case SC_Read:
  val = kernel->machine->ReadRegister(4);
  {
  char *buffer = &(kernel->machine->mainMemory[val]);
  size =kernel->machine->ReadRegister(5);
  id = kernel->machine->ReadRegister(6);
  status =SysRead(buffer,size,id);
  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;


case SC_Close:
  val = kernel->machine->ReadRegister(4);
  {
  char *buffer = &(kernel->machine->mainMemory[val]);
  size =kernel->machine->ReadRegister(5);
  id = kernel->machine->ReadRegister(6);
  status = SysClose(id);
  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;
```

從r4讀取要打開的file的addr.

從mainmemory獲取要打開的file的位址

執行sysOpen的結果存於status

將status中的結果寫到r2

set previous programm counter (debugging only)

將pc暫存器內的值+4寫回pc暫存器中(儲存下一條指令)

set next program counter for branch execution

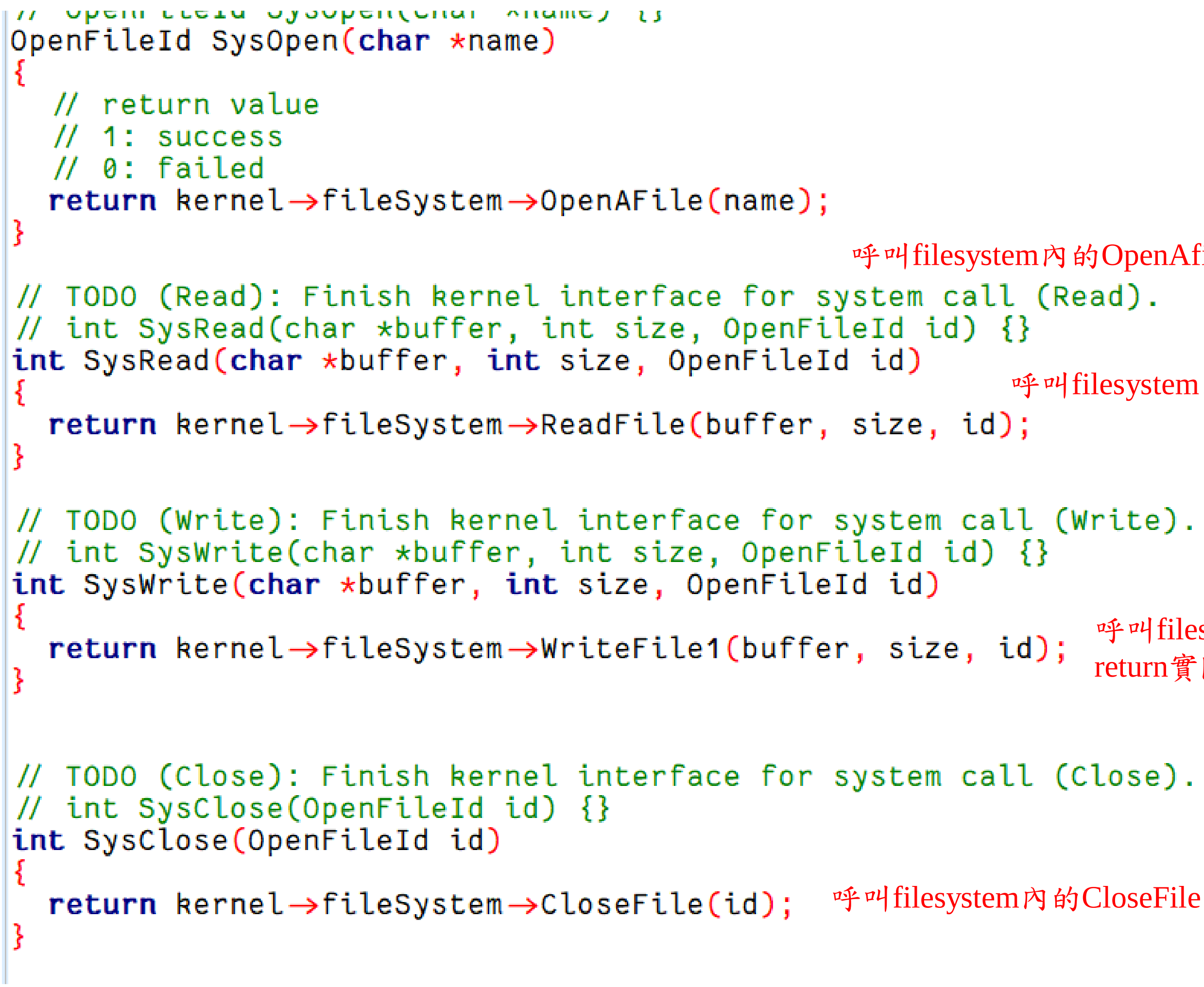
表示出現了錯誤

從r5讀取要寫入的內容大小

從r6中讀取要寫入的file descriptor

SysOpen需要去kernel的system call中處理，開啟userprog/ksyscall.h

SC_fileIO_test1 Trace code: userprog/ksyscall.h



```
// OpenFileId SysOpen(char *name) {}
OpenFileId SysOpen(char *name)
{
   // return value
   // 1: success
   // 0: failed
   return kernel->fileSystem->OpenAFile(name);
}

// TODO (Read): Finish kernel interface for system call (Read).
// int SysRead(char *buffer, int size, OpenFileId id) {}
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) {}
int SysWrite(char *buffer, int size, OpenFileId id)
{
   return kernel->fileSystem->WriteFile1(buffer, size, id);
}


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

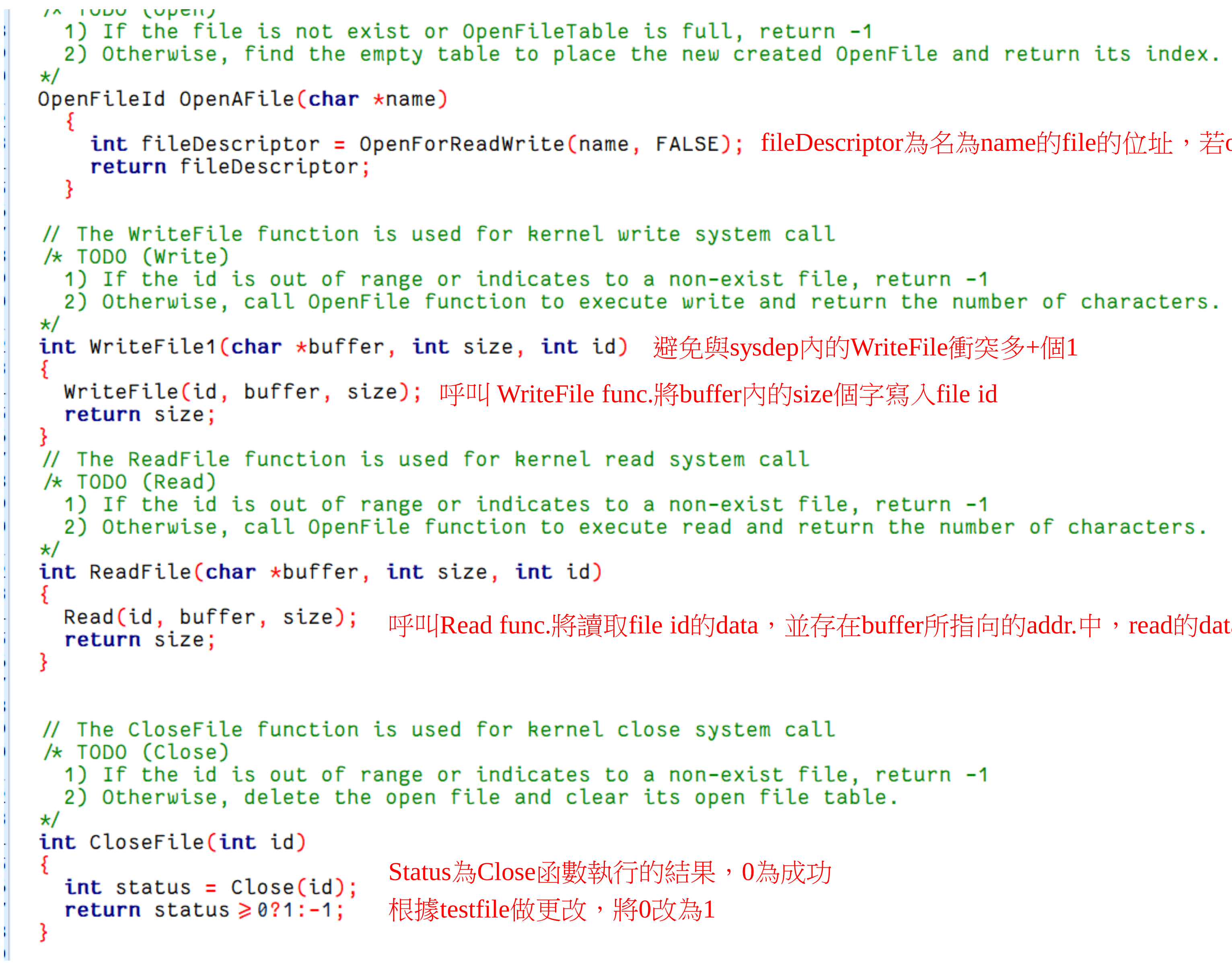
呼叫filesystem內的OpenAfile func. 並返回1or0代表成功失敗

呼叫filesystem內的ReadFile func.讀取指定file id中的size個字到buffer中，並return實際讀取的字數。

呼叫filesystem的WriteFile1 func.將buffer中size個字寫入指定file id中，並return實際寫入的字數。

呼叫filesystem內的CloseFile func關閉指定file id，並return 0表示成功

SC_fileIO_test1 Trace code: filesys/filesys.h



```
/* TODO (Open)
  1) If the file is not exist or OpenFileTable is full, return -1
  2) Otherwise, find the empty table to place the new created OpenFile and return its index.
*/
OpenFileId OpenAFile(char *name)
 {
   int fileDescriptor = OpenForReadWrite(name, FALSE);
   return fileDescriptor;
 }
```

fileDescriptor為名為name的file的位址，若open失敗則return False。

```
// The WriteFile function is used for kernel write system call
/* TODO (Write)
  1) If the id is out of range or indicates to a non-exist file, return -1
  2) Otherwise, call OpenFile function to execute write and return the number of characters.
*/
int WriteFile1(char *buffer, int size, int id)
{
  WriteFile(id, buffer, size);
  return size;
}
```

避免與sysdep內的WriteFile衝突多+個1

呼叫 WriteFile func.將buffer內的size個字寫入file id

```
// The ReadFile function is used for kernel read system call
/* TODO (Read)
  1) If the id is out of range or indicates to a non-exist file, return -1
  2) Otherwise, call OpenFile function to execute read and return the number of characters.
*/
int ReadFile(char *buffer, int size, int id)
{
  Read(id, buffer, size);
  return size;
}
```

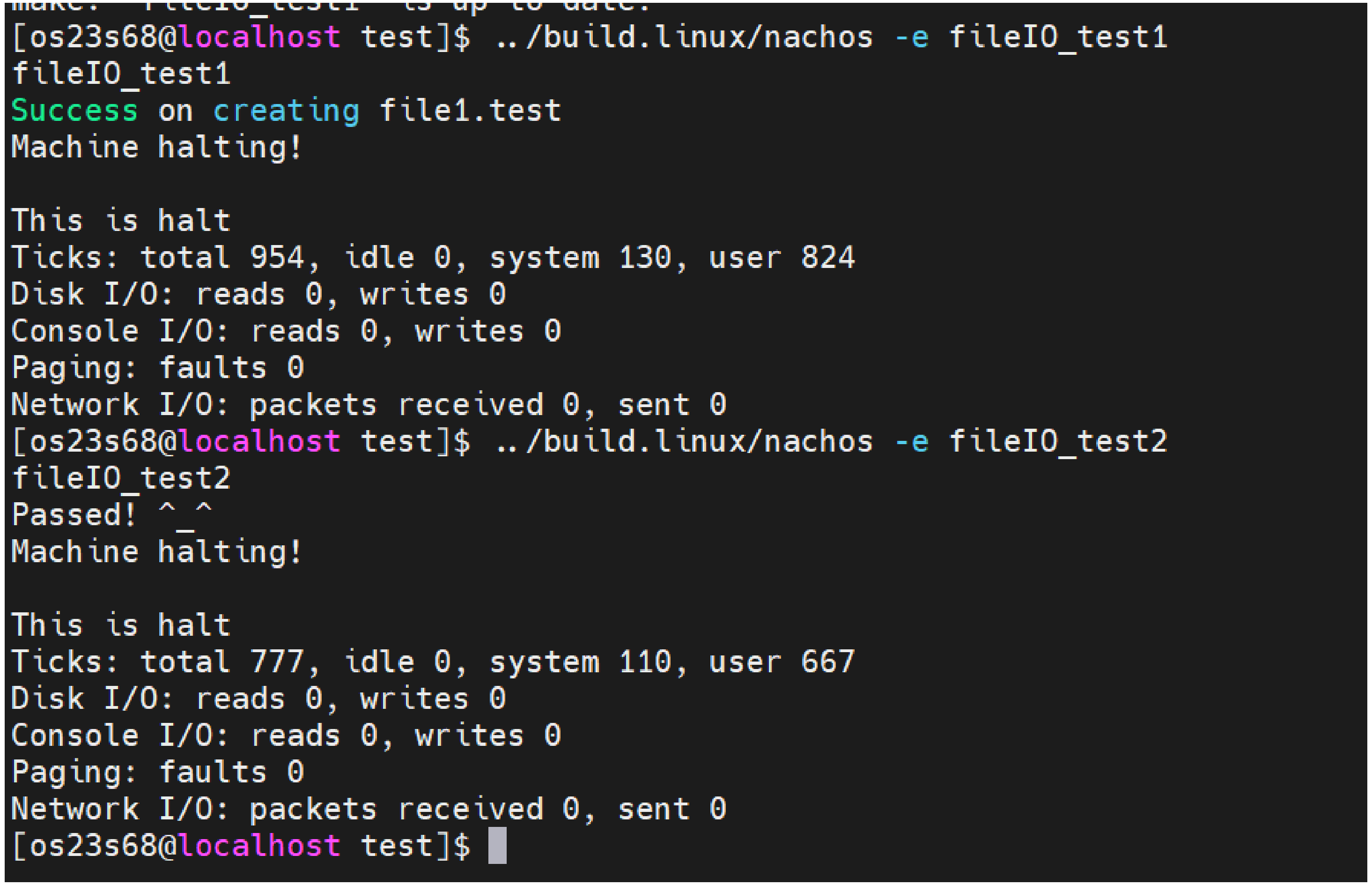
呼叫Read func.將讀取file id的data，並存在buffer所指向的addr.中，read的data長度為size

```
// The CloseFile function is used for kernel close system call
/* TODO (Close)
  1) If the id is out of range or indicates to a non-exist file, return -1
  2) Otherwise, delete the open file and clear its open file table.
*/
int CloseFile(int id)
{
  int status = Close(id);
  return status ≥0?1:-1;
}
```

Status為Close函數執行的結果，0為成功

根據testfile做更改，將0改為1

SC_fileIO_test1 Demo



```
make: `fileIO_test1' is up to date.
[os23s68@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
[os23s68@localhost test]$ ../build.linux/nachos -e fileIO_test2
fileIO_test2
Passed! ^_^
Machine halting!

This is halt
Ticks: total 777, idle 0, system 110, user 667
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
[os23s68@localhost test]$
```

What difficulties did you encounter when implementing this assignment?

由於對C++有點遺忘，因此花了些許時間去複習。

Any feedback you would like to let us know.

感謝助教提供TODO方便我去查詢哪些File是需要更改的。