OSTeam19_Report

Team_member：鍾昀諠，陳麒懋

Contribution：

鍾昀諠：Part1，code report

陳麒懋：Part2，trace report

Part I：Console I/O

Syscall.h

```
55  /* Print Integer */
56  #define SC_PrintInt 87
57  void PrintInt(int number);
```

因為 Syscall.h 只是整個 system call 的介面，所以在這裡只需要宣告要新增的 system call 函數和常數讓接下來的 exception.cc 用於判斷被呼叫的是哪一個 system call。

Start.S

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

這邊負責的是以 MIPS 指令把資料寫進 register 裡的動作，把$2 寫入代表這是哪個 system call 的常數。

Exception.cc

```
121         case SC_PrintInt:
122             DEBUG(dbgAddr, "Print Integer\n");
123             val = kernel->machine->ReadRegister(4);
124             cout << "Val = " << val << "\n";
125             SysPrintInt(val);
126             {
127             /* set previous programm counter (debugging only)*/
128             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
129 
130             /* set programm counter to next instruction (all Instructions are 4 byte wide)*/
131             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
132 
133             /* set next programm counter for brach execution */
134             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
135             }
136             return;
137             ASSERTNOTREACHED();
138             break;
```

Exception.cc 負責查看$2 裡面存放的是代表哪一個 system call 的常數，並把他 map 到對應的函數進行處理。這裡我們新增了處理 PrintInt 的 case，在裡面把要印出來的值從$4 取出來傳下去，並處理 Program Counter。

Ksyscall.h

```
37  void SysPrintInt(int number)
38  {
39      kernel->interrupt->PrintInt(number);
40  }
```

Exception.cc 把取出來要印的數字傳遞到 SysPrintInt 來，SysPrintInt 再把參數傳到 kernel 處理，因為要轉換成 kernel mode 執行，所以必須先傳到 interrupt 修改 user mode，再從 interrput 呼叫 kernel 來執行。

Interrupt.h/cc

```
99      void PrintInt(int number);
```

在.h 檔宣告函數，然後在.cc 實作，是比較好的一種 coding style。

```
248 void
249 Interrupt::PrintInt(int number)
250 {
251     kernel->PrintInt(number);
252 }
```

在 interrupt.cc 就可以直接呼叫 kernel 進行處理。

Kernel.h/cc

```
49      void PrintInt(int number);
```

宣告函數。

```
311 void Kernel::PrintInt(int number)
312 {
313     synchConsoleOut->PrintInt(number);
314 }
```

因為顯示設備是 asynchronous 的，所以不能直接 call console.cc 把他印出來，而是要先到 synchConsoleOut 去把 device 鎖起來不讓其他人使用，確保不會被干擾再印。

SynConsole.h/cc

```
45      void PrintInt(int number);
```

宣告函數

```
115 void
116 SynchConsoleOutput::PrintInt(int number)
117 {
118     lock->Acquire();
119     consoleOutput->PrintInt(number);
120     waitFor->P();
121     lock->Release();
122 }
```

第 118 行 call threads/synch.cc 裡面的 lock 函數把 output device 鎖起來，接著 119 行呼叫 consoleOutput 把數字輸出，120 行負責判斷 output 是否已經結束，121 行重新開放 device 供其他程式使用。

Console.h/cc

```
82      void PrintInt(int number);
```

```
175 void
176 ConsoleOutput::PrintInt(int number)
177 {
178     char ch[32] = {};
179     char c = '\n';
180     int size = sprintf(ch,"%d",number);
181     ASSERT(putBusy == FALSE);
182     for(int i=0;i<size;i++) WriteFile(writeFileNo, &ch[i], sizeof(char));
183     WriteFile(writeFileNo, &c, sizeof(char));
184     putBusy = TRUE;
185     kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
186 }
```

最後終於可以實作輸出數字的部分，先用 sprintf 把正負皆可的數字轉成由字元組成的 array，然後一個一個字元輸出，然後把這個 interrupt 放進 scheduler 放進執行序列阜

排程執行。

Part II：File I/O

Syscall.h

```
27  #define SC_Open     6
28  #define SC_Read     7
29  #define SC_Write    8
30  #define SC_Close    10
```

代表 system call 的常數，和 PartI 一樣是為了讓 OS 知道是哪一個 system call 被呼叫。

```
130 OpenFileId Open(char *name);
131
132 /* Write "size" bytes from "buffer" to the open file.
133  * Return the number of bytes actually read on success.
134  * On failure, a negative error code is returned.
135  */
136 int Write(char *buffer, int size, OpenFileId id);
137
138 /* Read "size" bytes from the open file into "buffer".
139  * Return the number of bytes actually read -- if the open file isn't
140  * long enough, or if it is an I/O device, and there aren't enough
141  * characters to read, return whatever is available (for I/O devices,
142  * you should always wait until you can return at least one character).
143  */
144 int Read(char *buffer, int size, OpenFileId id);
145
146 /* Close the file, we're done reading and writing to it.
147  * Return 1 on success, negative error code on failure
148  */
149 int Close(OpenFileId id);
```

Start.S

```
127  Open:
128      addiu $2,$0,SC_Open
129      syscall
130      j   $31
131      .end Open
132
133      .globl Read
134      .ent    Read
135  Read:
136      addiu $2,$0,SC_Read
137      syscall
138      j   $31
139      .end Read
140
141      .globl Write
142      .ent    Write
143  Write:
144      addiu $2,$0,SC_Write
145      syscall
146      j   $31
147      .end Write
148
149      .globl Close
150      .ent    Close
151  Close:
152      addiu $2,$0,SC_Close
153      syscall
154      j   $31
155      .end Close
```

把 syscall.h 定義的 syscall 常數放進$2 裡面。

Exception.cc

Open

```
139          case SC_Open:
140              DEBUG(dbgAddr,"Open File\n");
141              val = kernel->machine->ReadRegister(4);
142              {
143                  char *filename = &(kernel->machine->mainMemory[val]);
144                  status = SysOpen(filename);
145                  kernel->machine->WriteRegister(2, (int) status);
146              }
147              kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
148              kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
149              kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
150              return;
151              ASSERTNOTREACHED();
152              break;
```

把要開啟的 file 的位址從$4 取出來傳下去，並把開啟檔案的結果寫到$2，接著處理 PC 後 return 該檔案的 File Descriptor 或是代表失敗的-1。

Read

```
153         case SC_Read:
154             DEBUG(dbgAddr,"Read File\n");
155             val = kernel->machine->ReadRegister(4);
156             {
157                 char *buffer = &(kernel->machine->mainMemory[val]);
158                 // 5,6 ??
159                 int size = kernel->machine->ReadRegister(5);
160                 int id = kernel->machine->ReadRegister(6);
161                 status = SysRead(buffer,size,id);
162                 kernel->machine->WriteRegister(2, (int) status);
163             }
164             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
165             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
166             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
167             return;
168             ASSERTNOTREACHED();
169             break;
```

從 registers 讀出要寫入的 buffer 的位址、資料的大小以及要讀檔案的 fileID 並傳下去，回傳讀入的 byte 數並寫到 $2，接著處理 PC 後 return。

Write

```
170         case SC_Write:
171             DEBUG(dbgAddr,"Write File\n");
172             val = kernel->machine->ReadRegister(4);
173             {
174                 char *buffer = &(kernel->machine->mainMemory[val]);
175                 // 5 6 ??
176                 int size = kernel->machine->ReadRegister(5);
177                 int id = kernel->machine->ReadRegister(6);
178                 status = SysWrite(buffer,size,id);
179                 kernel->machine->WriteRegister(2, (int) status);
180             }
181             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
182             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
183             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
184             return;
185             ASSERTNOTREACHED();
186             break;
```

從 registers 讀出要寫入的檔案的位址、資料的大小以及要讀的 buffer 的 fileID 傳下去，回傳寫入的 byte 數並寫到 $2，接著處理 PC 後 return。

Close

```
187         case SC_Close:
188             DEBUG(dbgAddr,"Close File\n");
189             val = kernel->machine->ReadRegister(4);
190             {
191                 status = SysClose(val);
192                 kernel->machine->WriteRegister(2, (int) status);
193             }
194             kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
195             kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
196             kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);
197             return;
198             ASSERTNOTREACHED();
199             break;
```

從 registers 讀出要關掉的檔案的位址傳下去，如果成功關閉回傳 1，失敗回傳 0，並寫入到$2，接著處理 PC 後 return。

Ksyscall.h

```
42  OpenFileId SysOpen(char *filename)
43  {
44      kernel->interrupt->OpenFile(filename);
45  }
46
47  int SysRead(char *buffer, int size, OpenFileId id)
48  {
49      kernel->interrupt->Read(buffer,size,id);
50  }
51
52  int SysWrite(char *buffer, int size, OpenFileId id)
53  {
54      kernel->interrupt->Write(buffer,size,id);
55  }
56
57  int SysClose(OpenFileId id)
58  {
59      kernel->interrupt->Close(id);
60  }
```

Exception.cc 把取出來的參數傳遞到 Ksyscall.h 裡面來，Ksyscall.h 再把參數傳到 kernel 處理，因為要轉換到 kernel mode 執行，所以必須先 call interrupt 修改 user

mode，再從 interrput 呼叫 kernel 來執行。

Interrupt.h/cc

```
102     OpenFileId OpenFile(char *filename);
103     int Read(char *buffer, int size, OpenFileId id);
104     int Write(char *buffer, int size, OpenFileId id);
105     int Close(OpenFileId id);
```

宣告函數

```
254 OpenFileId Interrupt::OpenFile(char *filename)
255 {
256     return kernel->OpenFile(filename);
257 }
258
259 int Interrupt::Read(char *buffer, int size, OpenFileId id)
260 {
261     return kernel->Read(buffer,size,id);
262 }
263
264 int Interrupt::Write(char *buffer, int size, OpenFileId id)
265 {
266     return kernel->Write(buffer,size,id);
267 }
268
269 int Interrupt::Close(OpenFileId id)
270 {
271     return kernel->Close(id);
272 }
```

在 interrupt 因為已經把 mode 切換成 kernel mode，所以就可以直接 call kernel 執行。

Kernel.h/cc

```
50     OpenFileId OpenFile(char *filename);
51     int Read(char *buffer, int size, OpenFileId id);
52     int Write(char *buffer, int size, OpenFileId id);
53     int Close(OpenFileId id);
```

宣告函數

```
316  OpenFileId Kernel::OpenFile(char *filename)
317  {
318      int ID = (int)fileSystem->Open(filename);
319      if(ID!=0) return ID;
320      else return -1;
321  }
322
323  int Kernel::Read(char *buffer, int size, OpenFileId id)
324  {
325      return fileSystem->Read(buffer,size,id);
326  }
327
328  int Kernel::Write(char *buffer, int size, OpenFileId id)
329  {
330      return fileSystem->Write(buffer,size,id);
331  }
332
333  int Kernel::Close(OpenFileId id)
334  {
335      return fileSystem->Close(id);
336  }
```

Kernel 直接呼叫 filesystem，由 filesystem 裡面的函數去實作。注意在 OpenFile 中我們把傳回來的指標先轉換成 int 才傳回去。

Filesys.h

Open

```
57      OpenFile* Open(char *name) {
58        int fileDescriptor = OpenForReadWrite(name, FALSE);
59
60        if (fileDescriptor == -1) return NULL;
61
62        OpenFile* file = new OpenFile(fileDescriptor);
63        fileDescriptorTable[fileDescriptor] = file;
64        return file;
65        //return new OpenFile(fileDescriptor);
66        }
```

用 lib/sysdep 的 OpenForReadWrite 尋找檔案，如果沒有找到的話就回傳 null，找到的話就用其 index 創建一個 OpenFile 物件，接著以一個指標指向他，加入到 FileDescriptorTable 並回傳指標。

Read

```
68      int Read(char *buffer, int size, OpenFileId id)
69      {
70          int check,i = 0;
71          for(i=0;i<20;i++){
72              if(fileDescriptorTable[i]!=NULL && (OpenFileId)fileDescriptorTable[i] == id){
73                  check = 1;
74                  break;
75              }
76          }
77
78          if(check) return fileDescriptorTable[i]->Read(buffer,size);
79          else return -1;
80      }
```

用傳入的 FileID 在 FileDescriptorTable 中尋找想讀的檔案是否已經被開啟了。因為 table 裡面存的是指標，所以需要先做型別轉換才能跟 FileID 比較。接著 78 行用 openfile 物件內涵的 public 函數 Read 完成。

Write

```
82      int Write(char *buffer, int size, OpenFileId id)
83      {
84          int check,i = 0;
85          for(i=0;i<20;i++){
86              if(fileDescriptorTable[i]!=NULL && (OpenFileId)fileDescriptorTable[i] == id){
87                  check = 1;
88                  break;
89              }
90          }
91
92          if(check) return fileDescriptorTable[i]->Write(buffer,size);
93          else return -1;
94      }
```

用傳入的 FileID 在 FileDescriptorTable 中尋找我們想寫入資料的檔案是否已經被開啟了。因為 table 裡面存的是指標，所以需要先做型別轉換才能跟 FileID 比較。接著 92 行用 openfile 物件內涵的 public 函數 Write 完成。

Close

```
96      int Close(OpenFileId id)
97      {
98          int check,i = 0;
99          for(i=0;i<20;i++){
100             if(fileDescriptorTable[i]!=NULL && (OpenFileId)fileDescriptorTable[i] == id){
101                 check = 1;
102                 break;
103             }
104         }
105
106         if(check){
107             delete fileDescriptorTable[i];
108             fileDescriptorTable[i] = NULL;
109             return 1;
110         }
111         else return 0;
112     }
```

用傳入的 FileID 在 FileDescriptorTable 中尋找我們想關閉的檔案是否已經被開啟了，如果有找到就把 table 內的 OpenFile 物件 delete 掉，把 table 的值設成 null 然後回傳 1，如果沒有找到則回傳 0。因為 table 裡面存的是指標，所以需要先做型別轉換才能跟 FileID 比較。