NachOS – Memory Management

2024 / 12 / 03

Assignment

Assignment

- 在 NachOS-Lab3 的 code/test/ 路徑下有兩個檔案：

  consoleIO_test1.c (左)、consoleIO_test2.c (右)

```
#include "syscall.h"

main() {
     int n;
     for (n=9; n>5; n--) {
          PrintInt(n);
     }
     return ;
}
```

```
#include "syscall.h"

main() {
     int n;
     for (n=15; n<=19; n++) {
          PrintInt(n);
     }
     return ;
}
```

- consoleIO_test1.c 的執行結果 (理想上) 會將 9、8、7、6 依序印出
- consoleIO_test2.c 的執行結果 (理想上) 會將 15、16、17、18 、19 依序印出

Assignment

- 透過 NachOS 同時執行兩個檔案時會得到以下**錯誤**結果

```
05:30:29 root@74261e98c303 test → ../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2
consoleIO_test1
consoleIO_test2
9
16
15
18
19
^C
Cleaning up after signal 2
10:24:28 root@51c1a21d2138 test →
```

補充：因為在 consoleIO_test1 與 consoleIO_test2 未加入 Halt() Function，因此需要同學透過 control + c 中斷 NachOS 執行

Assignment

- **正確**結果應該如下

```
10:24:21 root@51c1a21d2138 test → ../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2
consoleIO_test1
consoleIO_test2
9
8
7
6
15
16
17
18
19
^C
Cleaning up after signal 2
10:24:28 root@51c1a21d2138 test →
```

補充：因為在 consoleIO_test1 與 consoleIO_test2 未加入 Halt() Function，因此需要同學透過 control + c 中斷 NachOS 執行

Problem

- 再觀察 Debug Message

```
== Tick 50 ==
      interrupts: on -> off
Time: 50, interrupts off
Pending interrupts:
Interrupt handler timer, scheduled at 100Interrupt handler console read,
End of pending interrupts
      interrupts: off -> on
Initializing address space: 12, 1536
Initializing code segment.
0, 400
Initializing stack pointer: 1520
== Tick 91 ==
      interrupts: on -> off
Time: 91, interrupts off
Pending interrupts:
Interrupt handler timer, scheduled at 100Interrupt handler console read,
181
End of pending interrupts
      interrupts: off -> on
Initializing address space: 12, 1536
Initializing code segment.
0, 400
Initializing stack pointer: 1520
```

Assignment

- NachOS-Lab3 目標

  解決 Page Table 實作不完全造成 Code 互相覆蓋的錯誤。

- Lab 要求 (需要實作的檔案)

  1. 助教已經在 machine/machine.h 新增新的 Exception 類型名為 MemoryLimitException，如果發生 Virtual Page 超過 Physical Page 的情境就該直接呼叫此 Exception (視為錯誤，結束 NachOS)。
  2. 在 **threads/kernel.*** 已經宣告並實作 Frame Table，Frame Table 的使用方式請自行 Trace Code。
  3. 在 **userprog/addrspace.cc - Addrspace::Addrspace()** 會需要將 page Table 初始化；**userprog/addrspace.cc - Addrspace::~Addrspace()** 需要將 Frame Table 已經被使用完畢的空間釋放。

Assignment

- Lab 要求

4. **userprog/addrspace.cc - bool AddrSpace::Load(char *fileName)** 需要實作多項機制

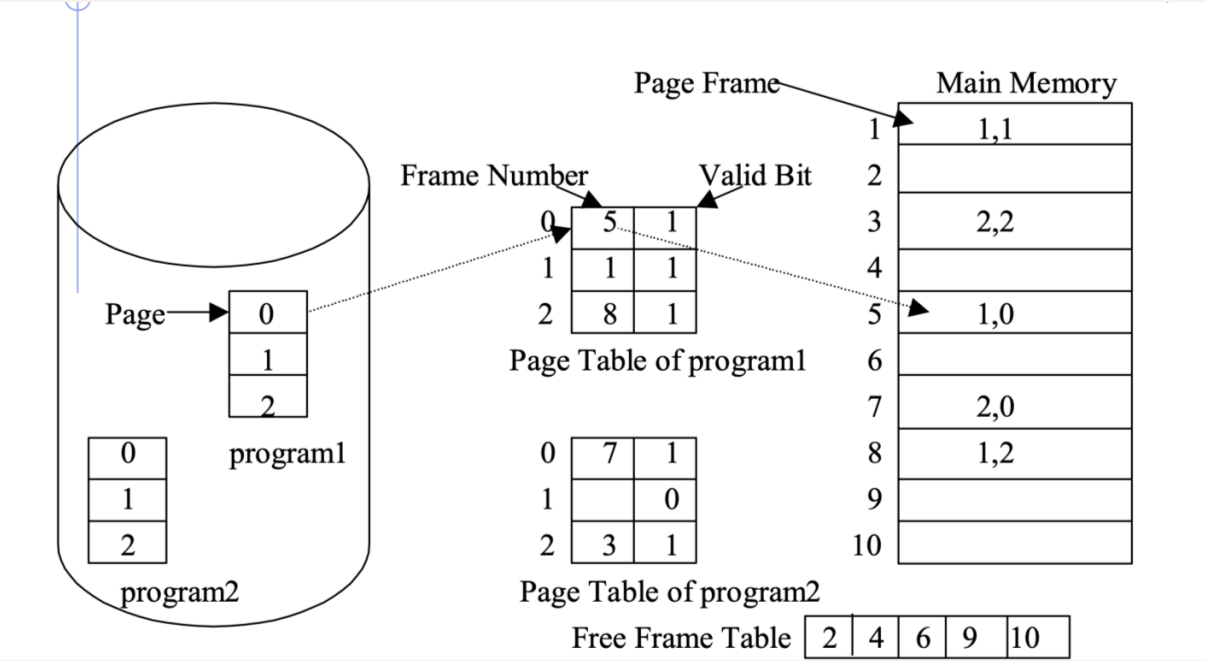
   - 檢查 Virtual Page index 是否超過 Physical Page index。

     如果發生此情境，則要呼叫 MemoryLimitException。

   - 尋找可使用的 Physical address，並在 page Table 紀錄其資訊。

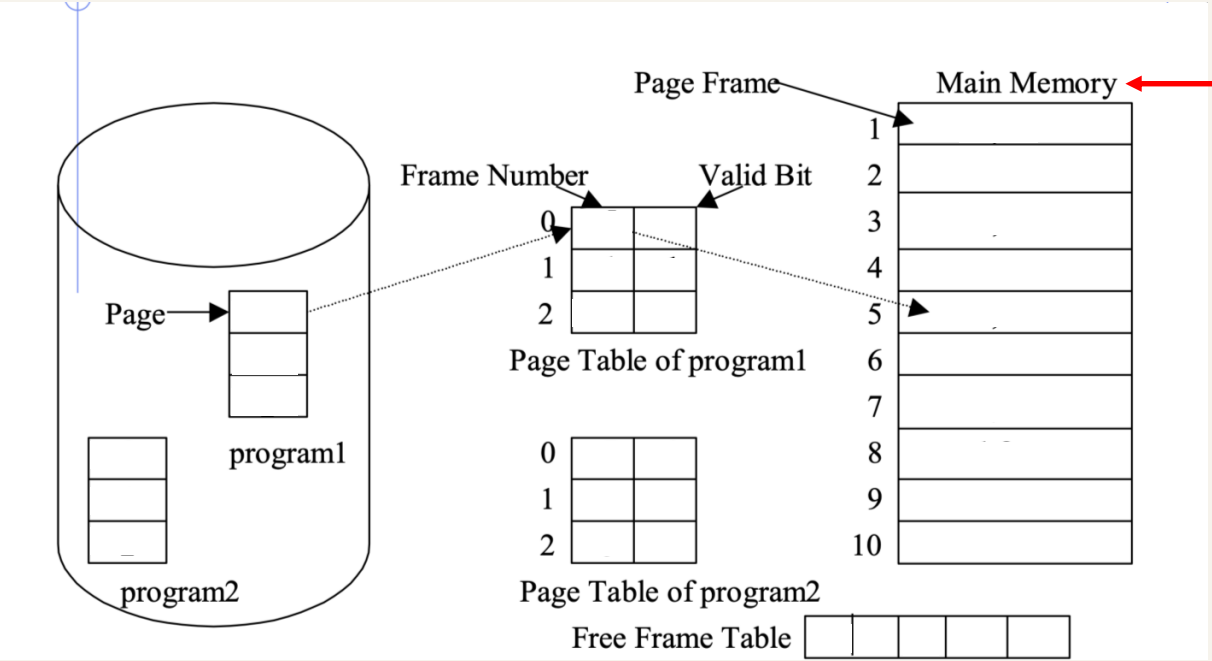
Assignment

- Page : 將程式依照固定大小進行切割。



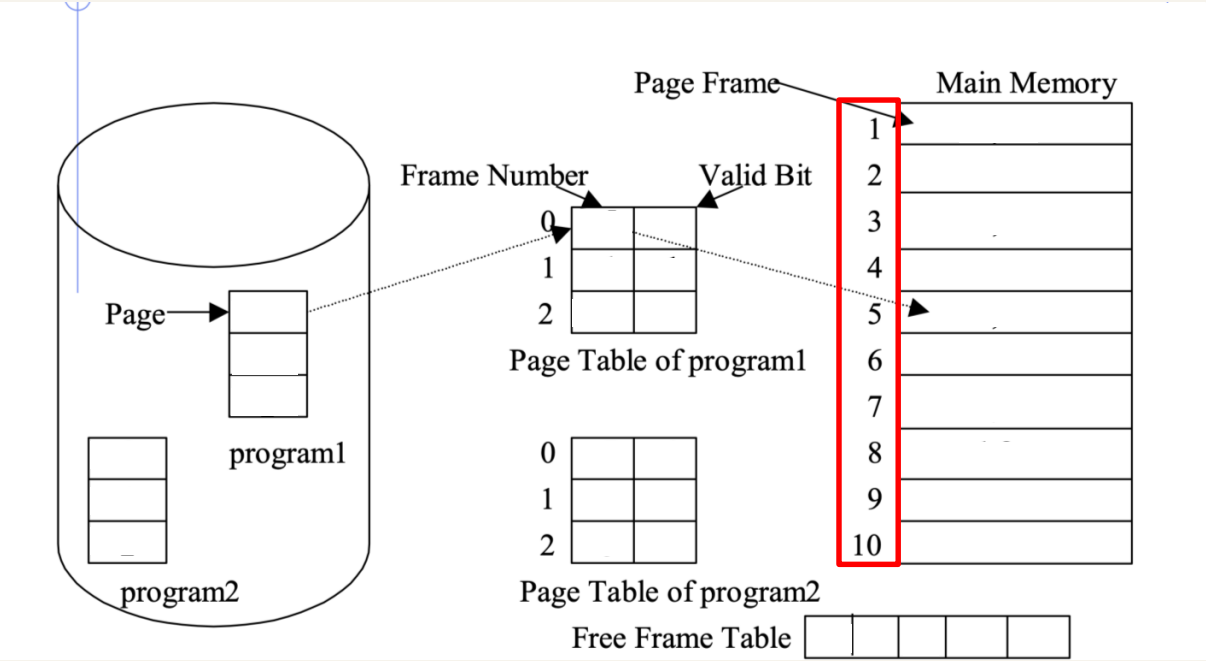
Assignment

- Physical Memory



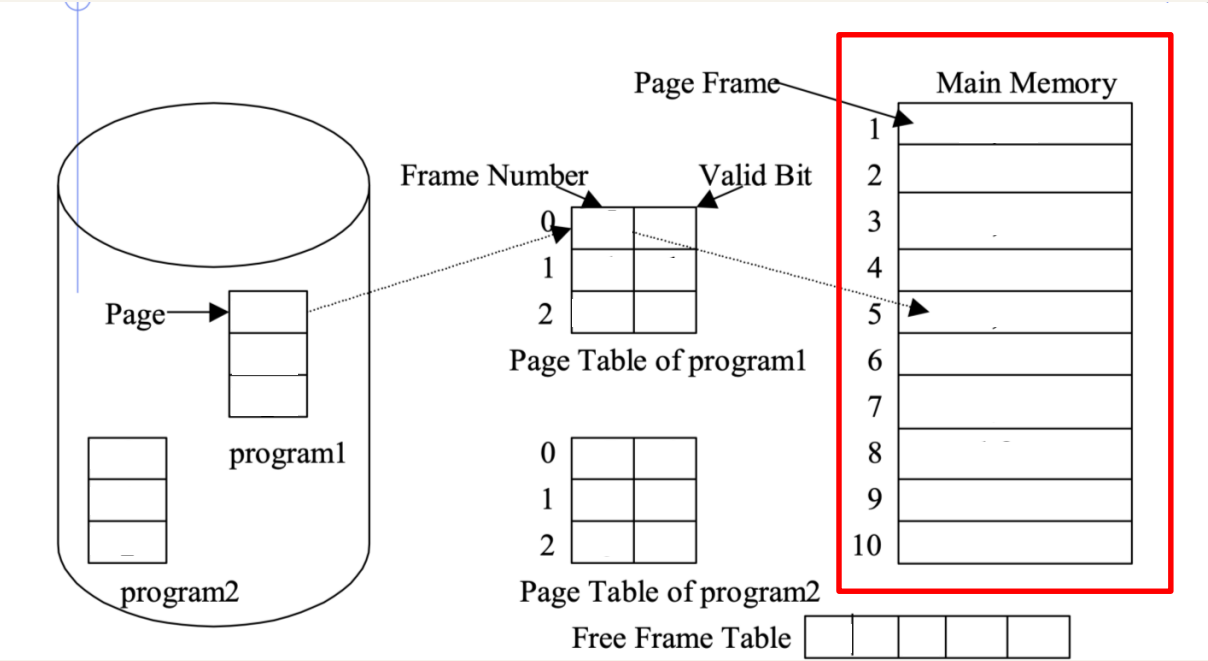
Assignment

- Physical Memory (const int NumPhysPages = 128 、int physicalPage)



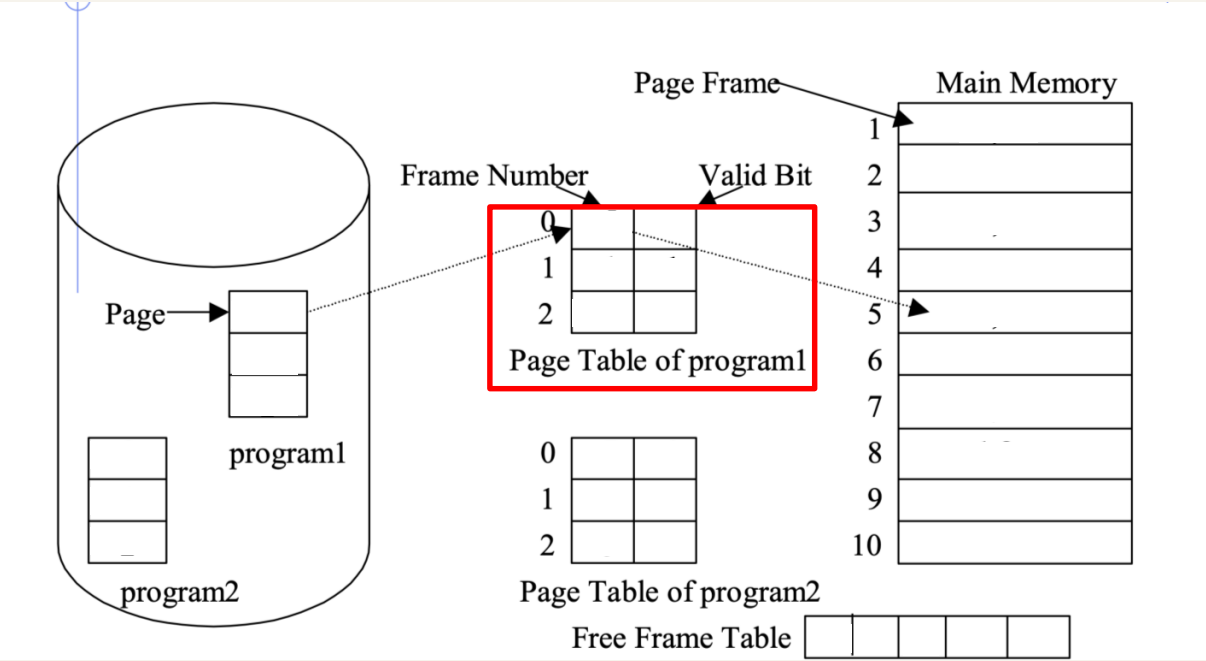
Assignment

- Frame Table (bool * frameTable)



Assignment

- Page Table (TranslationEntry *pageTable)



Assignment

- NachOS-Lab3 目標

  解決 Page Table 實作不完全造成 Code 互相覆蓋的錯誤。

- Lab 要求 (需要實作的檔案)

  1. 助教已經在 machine/machine.h 新增新的 Exception 類型名為 MemoryLimitException，如果發生 Virtual Page 超過 Physical Page 的情境就該直接呼叫此 Exception (視為錯誤，結束 NachOS)。
  2. 在 **threads/kernel.*** 已經宣告並實作 Frame Table，Frame Table 的使用方式請自行 Trace Code。
  3. 在 **userprog/addrspace.cc - Addrspace::Addrspace()** 會需要將 page Table 初始化；**userprog/addrspace.cc - Addrspace::~Addrspace()** 需要將 Frame Table 已經被使用完畢的空間釋放。

Assignment

- Lab 要求

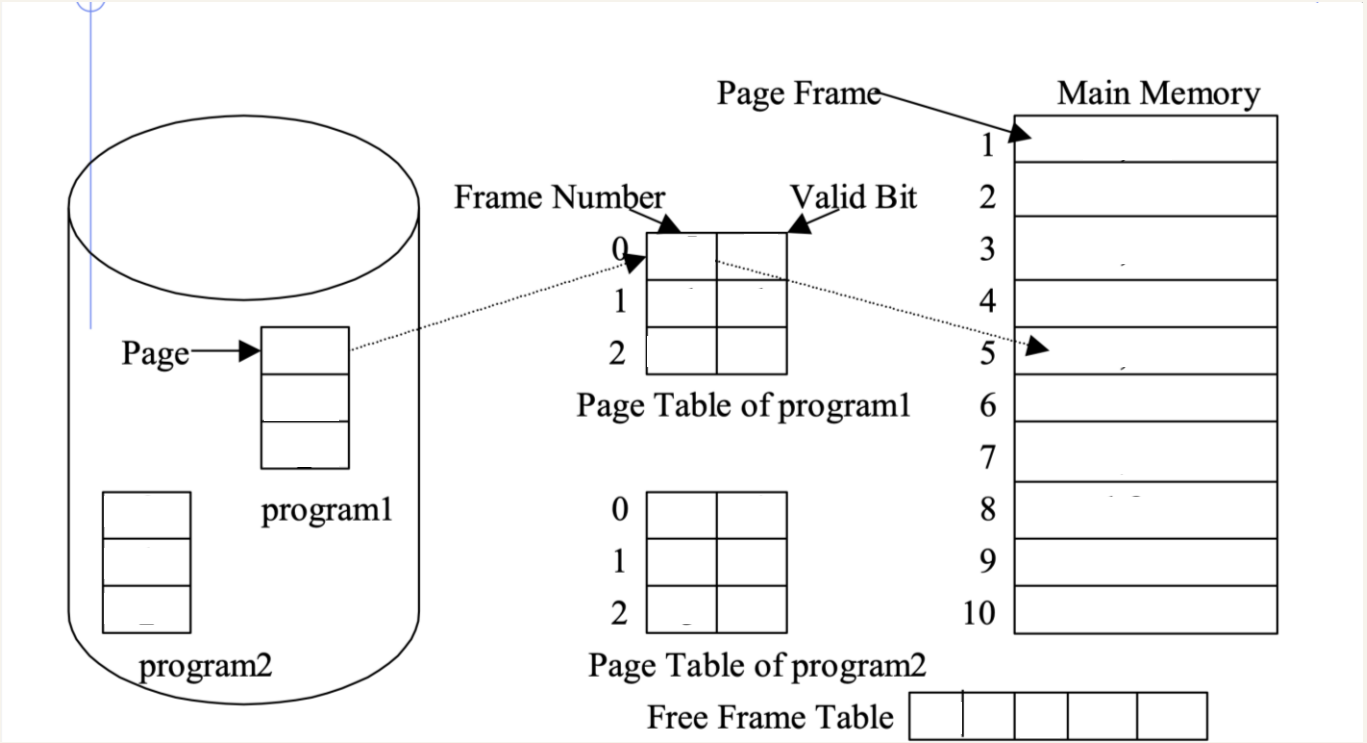
4. **userprog/addrspace.cc - bool AddrSpace::Load(char *fileName)** 需要實作多項機制

   - 檢查 Virtual Page index 是否超過 Physical Page index。

     如果發生此情境，則要呼叫 MemoryLimitException。

   - 尋找可使用的 Physical address，並在 page Table 紀錄其資訊。

Assignment


Page Frame
Main Memory
Frame Number
Valid Bit
Page
program1
program2
Page Table of program1
Page Table of program2
Free Frame Table
0
1
2
1
2
3
4
5
6
7
8
9
10

Hint - Trace Code

Trace Code

- 執行../build.linux/nachos -e consoleIO_test1 -e consoleIO_test2 發生了什麼事？

threads/main.cc
int main(int argc, char **argv)

threads/kernel.cc
Kernel::Kernel(int argc, char **argv)

threads/main.cc
int main(int argc, char **argv)

threads/kernel.cc
Kernel::ExecAll()
Kernel::Exec(char* name)

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec(char* name)
void ForkExecute(Thread *t)

userprog/addrspace.cc
bool AddrSpace::Load(char *fileName)

threads/main.cc
main()

threads/kernel.cc
Kernel::Kernel()

threads/main.cc
int main()

Trace Code

- threads/main.cc - **int main(int argc, char **argv)**
  - 上半部分在解譯使用者輸入的指令 (如：Lab2 的 -sche)。
  - 不過 -e 是在 new Kernel 時才進行解譯的。

```
247
248     DEBUG(dbgThread, "Entering main");
249
250     kernel = new Kernel(argc, argv);
251
252     kernel->Initialize();
253
254     CallOnUserAbort(Cleanup);       // if user hits ctl-C
255
```

threads/main.cc
main()

threads/kernel.cc
Kernel::Kernel()

threads/main.cc
int main()

Trace Code

- threads/kernel.cc - **Kernel::Kernel(int argc, char **argv)**
  - 將檔名以字串的形式存至 execfile[]
  - 詳見 threads/kernel.h

```
39      for (int i = 1; i < argc; i++) {
40          if (strcmp(argv[i], "-rs") == 0) {
41              ASSERT(i + 1 < argc);
42              RandomInit(atoi(argv[i + 1]));// initialize pseudo-random
43              // number generator
44              randomSlice = TRUE;
45              i++;
46          } else if (strcmp(argv[i], "-s") == 0) {
47              debugUserProg = TRUE;
48          } else if (strcmp(argv[i], "-e") == 0) {
49              execfile[++execfileNum]= argv[++i];
50              cout << execfile[execfileNum] << "\n";
```

Trace Code

threads/main.cc
main()

threads/kernel.cc
Kernel::Kernel()

threads/main.cc
int main()

- threads/main.cc - **int main(int argc, char **argv)**

```
247
248     DEBUG(dbgThread, "Entering main");
249
250     kernel = new Kernel(argc, argv);
251
252     kernel->Initialize();
253
254     CallOnUserAbort(Cleanup);       // if user hits ctl-C
255
```

threads/kernel.cc
Kernel::Kernel()

threads/main.cc
int main()

threads/kernel.cc
Kernel::ExecAll()
Kernel::Exec()

Trace Code

- threads/main.cc - **int main(int argc, char **argv)**
  - 進入 ExecAll()

```
286     // finally, run an initial user program if requested to do so
287
288         kernel->ExecAll();
289     // If we don't run a user program, we may get here.
290     // Calling "return" would terminate the program.
291     // Instead, call Halt, which will first clean up, then
292     //  terminate.
293 //    kernel->interrupt->Halt();
294
```

threads/kernel.cc
Kernel::Kernel()

threads/main.cc
int main()

threads/kernel.cc
Kernel::ExecAll()
Kernel::Exec()

Trace Code

- threads/kernel.cc - **Kernel::ExecAll()**
  - execfile[] 存放了需要運行的 Program，透過 for() 呼叫 Exec()

```
263
264  void Kernel::ExecAll()
265  {
266      for (int i=1;i<=execfileNum;i++) {
267          int a = Exec(execfile[i]);
268      }
269      currentThread->Finish();
270      //Kernel::Exec();
271  }
272
```

threads/kernel.cc
Kernel::Kernel()

threads/main.cc
int main()

threads/kernel.cc
Kernel::ExecAll()
Kernel::Exec()

Trace Code

- threads/kernel.cc - **Exec(char* name)**
  - 建立新的 Thread，並且透過 AddrSpace::AddrSpace() 初始化 space 成員。

    space 是 Thread 的成員，型別是 AddrSpace *space。

```
274  int Kernel::Exec(char* name)
275  {
276      t[threadNum] = new Thread(name, threadNum);
277      t[threadNum]->space = new AddrSpace();
278      t[threadNum]->Fork((VoidFunctionPtr) &ForkExecute, (void *)t[threadNum]);
279      threadNum++;
280
281      return threadNum-1;
```

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec()
void ForkExecute()

userprog/addrspace.cc
bool AddrSpace::Load()

Trace Code

- userprog/addrspace.cc - **AddrSpace::AddrSpace()**
  - AddrSpace::AddrSpace() 應該要做的事情：新增 pageTable、將 pageTable 的資訊進行初始化

    系統所提供的 code 未必正確，也不一定是錯的，取決於實作方式。

```
64
65  AddrSpace::AddrSpace() {
66      pageTable = new TranslationEntry[NumPhysPages];
67      for (int i = 0; i < NumPhysPages; i++) {
68          pageTable[i].virtualPage = i;  // for now, virt page # = phys page #
69          pageTable[i].physicalPage = i;
70          pageTable[i].valid = TRUE;
71          pageTable[i].use = FALSE;
72          pageTable[i].dirty = FALSE;
73          pageTable[i].readOnly = FALSE;
74      }
75
76      // zero out the entire address space
77      bzero(kernel->machine->mainMemory, MemorySize);
78  }
79
```

Trace Code

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec()
void ForkExecute()

userprog/addrspace.cc
bool AddrSpace::Load()

- threads/kernel.cc - **Exec(char* name)**
  - 透過 Thread::Fork() 將當前 Thread 以 Fork 的形式執行。

```
274  int Kernel::Exec(char* name)
275  {
276      t[threadNum] = new Thread(name, threadNum);
277      t[threadNum]->space = new AddrSpace();
278      t[threadNum]->Fork((VoidFunctionPtr) &ForkExecute, (void *)t[threadNum]);
279      threadNum++;
280
281      return threadNum-1;
```

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec()
void ForkExecute()

userprog/addrspace.cc
bool AddrSpace::Load()

Trace Code

- threads/kernel.cc - **void ForkExecute(Thread *t)**
  - 在前面有提到 space 成員的型別是 AddrSpace *space
    也因此 Load() 函式就要到userprog/addrspace.cc 尋找。

```
254  void ForkExecute(Thread *t)
255  {
256      if ( !t->space->Load(t->getName()) ) {
257          return;             // executable not found
258      }
259
260      t->space->Execute(t->getName());
261
262  }
263
```

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec()
void ForkExecute()

userprog/addrspace.cc
bool AddrSpace::Load()

Trace Code

- userprog/addrspace.cc - **bool AddrSpace::Load(char *fileName)**

  - Load() 函式的功能：**開啟檔案**、計算程式所需的 Virtual Address Space、將 code, initData, readonlyData 區段放入 Main Memory、關閉執行檔並回傳成功執行。

```
 99  bool AddrSpace::Load(char *fileName) {
100      OpenFile *executable = kernel->fileSystem->Open(fileName);
101      NoffHeader noffH;
102      unsigned int size;
103
104      if (executable == NULL) {
105          cerr << "Unable to open file " << fileName << "\n";
106          return FALSE;
107      }
```

Trace Code

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec()
void ForkExecute()

userprog/addrspace.cc
bool AddrSpace::Load()

- userprog/addrspace.cc - **bool AddrSpace::Load(char *fileName)**

  - Load() 函式的功能：開啟檔案、**計算程式所需的 Virtual Address Space**、將 code, initData, readonlyData 區段放入 Main Memory、關閉執行檔並回傳成功執行。

```
115   #ifdef RDATA
116       // how big is address space?
117       size = noffH.code.size + noffH.readonlyData.size + noffH.initData.size +
118              noffH.uninitData.size + UserStackSize;
119       // we need to increase the size
120       // to leave room for the stack
121   #else
122       // how big is address space?
123       size = noffH.code.size + noffH.initData.size + noffH.uninitData.size + UserStackSize;  // we need to increase the size
124                                                                                              // to leave room for the stack
125   #endif
126       numPages = divRoundUp(size, PageSize);
127       size = numPages * PageSize;
```

userprog/addrspace.cc
AddrSpace::AddrSpace()

threads/kernel.cc
Kernel::Exec()
void ForkExecute()

userprog/addrspace.cc
bool AddrSpace::Load()

Trace Code

- userprog/addrspace.cc - **bool AddrSpace::Load(char *fileName)**

  - Load() 函式的功能：開啟檔案、計算程式所需的 Virtual Address Space、**將 code, initData, readonlyData 區段放入 Main Memory**、關閉執行檔並回傳成功執行。

```
136     // then, copy in the code and data segments into memory
137     // Note: this code assumes that virtual address = physical address
138     if (noffH.code.size > 0) {
139         DEBUG(dbgAddr, "Initializing code segment.");
140         DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
141         executable->ReadAt(
142             &(kernel->machine->mainMemory[noffH.code.virtualAddr]),
143             noffH.code.size, noffH.code.inFileAddr);
144     }
```

END