HW2 – CPU Scheduler（補）

分工：

B10615045 陳尚富: 主要撰寫

B10601019 鞠傳豐: 二退

B10601033 陳彥憑: 負責 SJF

想法:

本次作業主要更改的檔案有:

Nachos/code/threads/scheduler.cc (.h)

Nachos/code/threads/threads.cc (.h)

Nachos/code/userprog/addrspace.cc (.h)

Nachos/code/threads/alarm.cc (.h)

Requirements 1:

在經過一段時間的 trace 之後，我發現原本 nachos 的記憶體配置上，設置 physical page table 和 virtual page table 是一對一的對應關係，一個 virtual 就直 接佔滿了整個 physical page table。

由於 addrspace 在執行 Execute 後，會先進行 Load 檢查，所以我就將有關這方面的實作寫在 Load 裡面。利用以經計算好的 numPages，就可以知道一個 thread 需要用多少的 page，這樣我就只需要在 physical page table 上，對應足夠 的大小給他即可，就不用配置整張表浪費。

接著我在配置 virtual page table 時，我就需要紀錄我已經使用了哪些 physical page table，所以我新增了一個 static 矩陣 physPageTable[]，儲存布林 值。還有一個變數 numFreePhyPages，用來記錄還剩下多少的空頁可以使用， 假如我得空頁不夠的話，我的 ASSERT 就會 Abort()。

```
    ASSERT(noffH.noffMagic == NOFFMAGIC);

// how big is address space?
    size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
            + UserStackSize;    // we need to increase the size
                                // to leave room for the stack
    numPages = divRoundUp(size, PageSize); // Number of pages used in this addr
//  cout << "number of pages of " << fileName<< " is "<<numPages<<endl;
    size = numPages * PageSize;

// @shungfu : Edit at Hw2
    ASSERT(numPages <= NumPhysPages);  // check we're not trying
                                       // to run anything bigger
                                       // than number of  physical pages
                                       // left in physicalPageTable
    DEBUG(dbgAddr, "Used " << numPages << "Pages")

//@shungfu : Edit at Hw2
    // Deploy the virtual and physical page tables.
    pageTable = new TranslationEntry[numPages];
    for(unsigned int i = 0; i < numPages; i++){
        // Deploy on only available index
        int j = i;
        while(j < NumPhysPages && physPageTable[j] == USED){
            j++;
        }

        physPageTable[j] = USED;
        pageTable[i].physicalPage = j;
        numFreePhyPages--;
        // Same as original
        pageTable[i].virtualPage = i;
        pageTable[i].valid = TRUE;
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;
    }

    DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);
```

Ubuntu 14.04.6

最後將 code 塞入 memory 的部分，我上網找到了方法如下:

- page base: which page * PageSize.
- page offset: code.address mod PageSize。
- mainMemory[]: page base + page offset

➔ mainMemory[pageTable[noffH.code.virtualAddr/PageSize].physicalPage * PageSize + (noffH.code.virtualAddr%PageSize)]

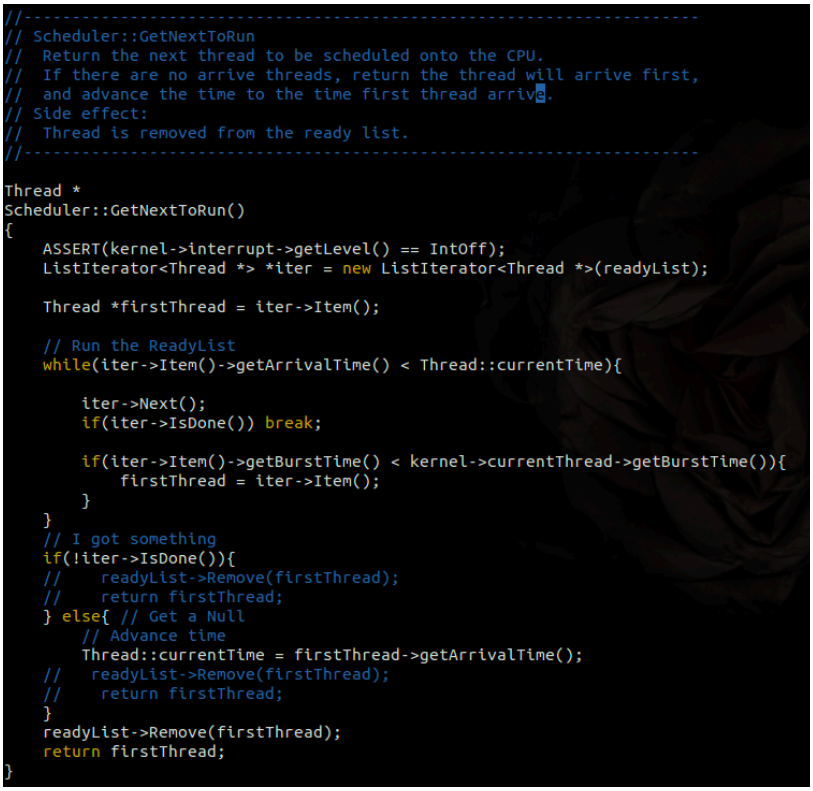
執行結果：

```
shungfu@ubuntu:~/NachOS/code$ ./userprog/nachos -e test/test1 -e test/test2
Total threads number is 2
Thread test/test1 is executing.
Thread test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:6
return value:0
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 300, idle 8, system 70, user 222
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
```

Requirements 2:

下圖是修改前。



```
//----------------------------------------------------------------------
// Scheduler::GetNextToRun
//  Return the next thread to be scheduled onto the CPU.
//  If there are no arrive threads, return the thread will arrive first,
//  and advance the time to the time first thread arrive.
// Side effect:
//  Thread is removed from the ready list.
//----------------------------------------------------------------------

Thread *
Scheduler::GetNextToRun()
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    ListIterator<Thread *> *iter = new ListIterator<Thread *>(readyList);

    Thread *firstThread = iter->Item();

    // Run the ReadyList
    while(iter->Item()->getArrivalTime() < Thread::currentTime){

        iter->Next();
        if(iter->IsDone()) break;

        if(iter->Item()->getBurstTime() < kernel->currentThread->getBurstTime()){
            firstThread = iter->Item();
        }
    }
    // I got something
    if(!iter->IsDone()){
    //    readyList->Remove(firstThread);
    //    return firstThread;
    } else{ // Get a Null
        // Advance time
        Thread::currentTime = firstThread->getArrivalTime();
    //   readyList->Remove(firstThread);
    //    return firstThread;
    }
    readyList->Remove(firstThread);
    return firstThread;
}
```

在這邊發現，我在 while 迴圈的部分寫錯了。我在排序方面是先依照 arrival time 的優先排序才依照 burst time 的優先排序，所以在尋找下一個應該要執行的對象，應該是要從比 current time 還要早到的 thread 中尋找 burst time 最小的，才是 SJF 和 SRTF 需要的。所以不應該是把 ready list 中的 thread 跟 current thread 比。而且如果現在沒有已經到達的 thread 就把 current 加快。然後這邊我覺得如果說我 arrival time 是 3 的時候，在 current time 為 3 的時候就應該進入，所以我也將 166 的<改為<= 。

```
156 Thread *
157 Scheduler::GetNextToRun()
158 {
159     ASSERT(kernel->interrupt->getLevel() == IntOff);
160     ListIterator<Thread *> *iter = new ListIterator<Thread *>(readyList);
161 
162     Thread *firstThread = iter->Item();
163     bool Found = false;
164 
165     // Run the ReadyList
166     while(iter->Item()->getArrivalTime() <= Thread::currentTime){
167         Found = true;
168         if(iter->Item()->getBurstTime() < firstThread->getBurstTime()){
169             firstThread = iter->Item();
170         }
171 
172         iter->Next();
173         if(iter->IsDone()) break;
174     }
175 
176     if(!Found){ // No thread is arrived
177         // Advance time
178         DEBUG(dbgHw2,"Advance :"  << firstThread->getName() << endl);
179         Thread::currentTime = firstThread->getArrivalTime();
180     }
181 
182     readyList->Remove(firstThread);
183     return firstThread;
184 }
```

SimpleThread()也做了改動，如果執行的是 SRTF 就要每次都先去 Yield，找到應該執行的 thread。

```
422 static void
423 SimpleThread()
424 {
425     Thread *thread = kernel->currentThread;
426     while (thread->getBurstTime() > 0) {
427 
428 
429         if(kernel->scheduler->getSchedulerType() == SRTF)
430             kernel->currentThread->Yield(); // will find next to run
431         thread->setBurstTime(thread->getBurstTime() - 1);
432 
433         DEBUG(dbgHw2, "Current time: " << Thread::currentTime);
434         printf("%s: %d\n", kernel->currentThread->getName(), kernel->currentThread->getBurstTime());
435 
436         kernel->interrupt->OneTick();
437         Thread::currentTime++;  // burst
438     }
439 }
```

測試程式：

```
void
Thread::SelfTest()
{
    DEBUG(dbgThread, "Entering Thread::SelfTest");

    const int number     = 5;
    char *name[number]   = {"A", "B", "C", "D", "E"};
    int burst[number]    = {1, 2, 3, 2, 1};
    int priority[number] = {4, 5, 3, 1, 2};
    int arrival[number] = {1, 1, 2, 3, 4};

    Thread *t;
    for (int i = 0; i < number; i++) {
        t = new Thread(name[i]);
        t->setPriority(priority[i]);
        t->setBurstTime(burst[i]);
        t->setArrivalTime(arrival[i]);
        t->Fork((VoidFunctionPtr) SimpleThread, (void *)NULL);
    }
    kernel->scheduler->Print();
    cout << endl;
    //kernel->currentThread->Yield();
}
```

修改前的結果圖：

```
shungfu@ubuntu:~/NachOS/code$ ./threads/nachos -SJF
Ready list contents:
ABCDE
A: 0
B: 1
B: 0
C: 2
C: 1
C: 0
D: 1
D: 0
E: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2400, idle 100, system 2300, user 0
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
```

```
shungfu@ubuntu:~/NachOS/code$ ./threads/nachos -SRTF
Ready list contents:
ABCDE
A: 0
B: 1
B: 0
D: 1
D: 0
C: 2
C: 1
C: 0
E: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2500, idle 80, system 2420, user 0
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
```

評語回饋

1.SJF 如果按照你報告的設定，輸出應該是要 A->B->D->E->C 輸出才對，結果與預期不符。

2.SRTF 輸出應該也會跟 SJF 一樣才對，結果圖中 E 的 burst time 只有 1，到達時間是 4，不應該是排在 C 後面，因為 C 的 burst time 是 3。

修改後的結果圖：

```
~/NachOS/code/threads  master ●  INSERT  ./nachos -SJF
Ready list contents:
ABCDE
A: 0
B: 1
B: 0
E: 0
D: 1
D: 0
C: 2
C: 1
C: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2400, idle 100, system 2300, user 0
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
```

```
~/NachOS/code/threads  master ●  INSERT  ./nachos -SRTF
Ready list contents:
ABCDE
A: 0
B: 1
B: 0
E: 0
D: 1
D: 0
C: 2
C: 1
C: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2600, idle 80, system 2520, user 0
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
```

但是經過修改後，結果應該是 A->B->E->D->C，因為時間點 4 的時候，E 就可以

arrive 了，又 E 的 Burst time 最小。

測試程式 2:

```
446 void
447 Thread::SelfTest()
448 {
449     DEBUG(dbgThread, "Entering Thread::SelfTest");
450
451     const int number     = 5;
452     char *name[number]   = {"A", "B", "C", "D", "E"};
453     int burst[number]    = {1, 7, 2, 7, 1};
454     int priority[number] = {4, 5, 3, 1, 2};
455     int arrival[number] = {1, 1, 3, 4, 5};
456
457     Thread *t;
458     for (int i = 0; i < number; i++) {
459         t = new Thread(name[i]);
460         t->setPriority(priority[i]);
461         t->setBurstTime(burst[i]);
462         t->setArrivalTime(arrival[i]);
463         t->Fork((VoidFunctionPtr) SimpleThread, (void *)NULL);
464     }
465     kernel->scheduler->Print();
466     cout << endl;
467     //kernel->currentThread->Yield();
468 }
```

結果：

```
~/NachOS/code/threads  master  INSERT  ./nachos -SJF
Ready list contents:
ABCDE
A: 0
B: 6
B: 5
B: 4
B: 3
B: 2
B: 1
B: 0
E: 0
C: 1
C: 0
D: 6
D: 5
D: 4
D: 3
D: 2
D: 1
D: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2400, idle 10, system 2390, user 0
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
```

```
~/NachOS/code/threads  master  INSERT  ./nachos -SRTF
Ready list contents:
ABCDE
A: 0
B: 6
C: 1
C: 0
E: 0
B: 5
B: 4
B: 3
B: 2
B: 1
B: 0
D: 6
D: 5
D: 4
D: 3
D: 2
D: 1
D: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2800, idle 80, system 2720, user 0
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
```

SRTF，在執行到時間點 3 的時候，C 中斷並且執行，時間點 5，C 結束並且 E 進入，E 結束後 B 的 burst time 最少，所以 B 繼續執行，最後 D 才執行到結束。