執行：

因為兩個任務合併時出現了一些 error 沒有被查出來，所以提交了兩份 code，分別對應任務 1 和任務 2。

目錄：

```
t08902109_Nachos2/
|_ nachos-4.0-1/
|_ nachos-4.0-2/
|_ report.pdf
```

System Call：

```
~ $cd ./t08902109_Nachos2/nachos-4.0-1/code
~/code $ make
~/code cd userprog
~/userprog $./nachos -e ./../test/test
```

CPU Scheduling:

```
~ $cd ./t08902109_Nachos2/nachos-4.0-2/code
~/code $ make
~/code cd threads
~/thread $./nachos FCFS
~/thread $./nachos SJF
~/thread $ ./nachos PRIORITY
~/thread $ ./nachos RR
```

一． System Call

1. Motivation

目標：撰寫 Sleep function，將 Thread 進入休眠。

計劃：

一方面，當程序呼叫 Sleep() 時，會呼叫 WaitUntil()，然後將其丟入 Bedroom 安眠。實現休眠。

另一方面，kernel 存有 alarm，每隔固定一段時間，就會呼叫 Alarm::CallBack()，因此，對於這個鬧鐘來個累加器_current_interrupt，全局去記數，每加一次就相當於過了 1 毫秒 (ms)。然後在 CallBlack() 被呼叫時，來去檢查誰該醒來。

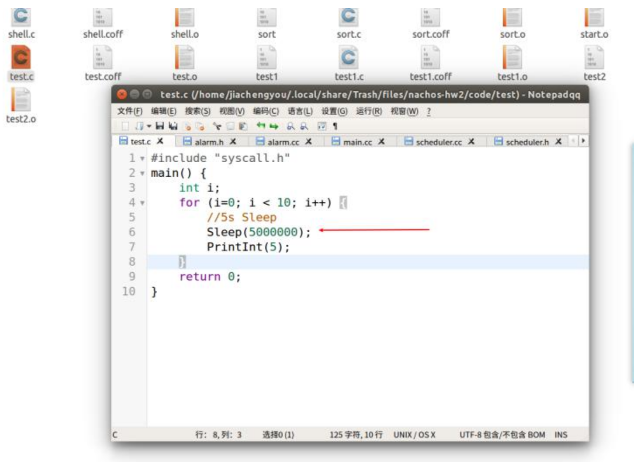
2. Implementation

執行 ：make

./nachos -e ./../test/test

可以看到休眠結果。

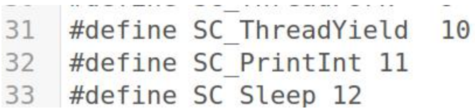
（1）Test function



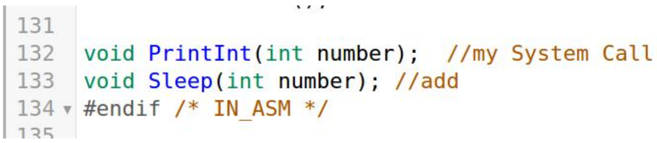
```c
#include "syscall.h"
main() {
    int i;
    for (i=0; i < 10; i++) {
        //5s Sleep
        Sleep(5000000);
        PrintInt(5);
    }
    return 0;
}
```

每五秒輸出一個整數 5

（2）準備 call code，宣告函數。

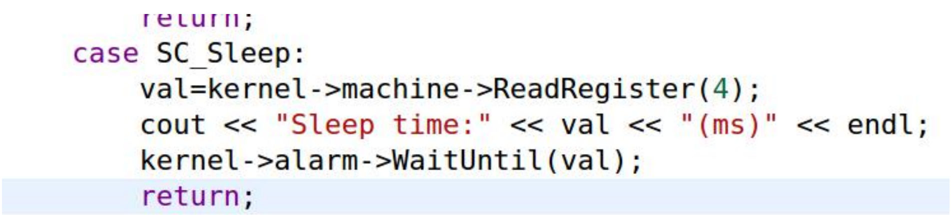


```c
#define SC_ThreadYield  10
#define SC_PrintInt 11
#define SC_Sleep 12
```



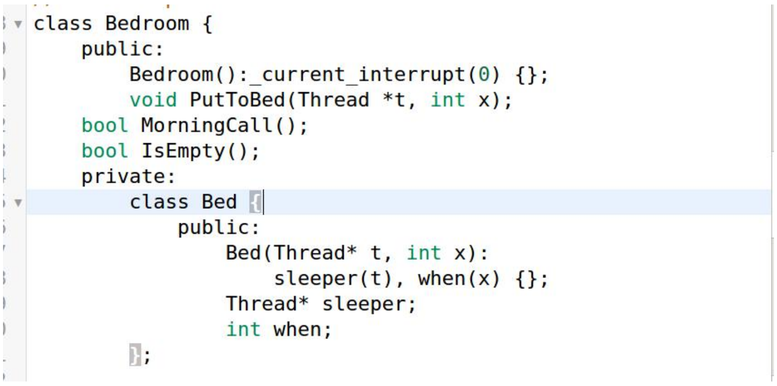
```c
void PrintInt(int number);  //my System Call
void Sleep(int number); //add
#endif /* IN_ASM */
```

（3）呼叫 WaitUntil()



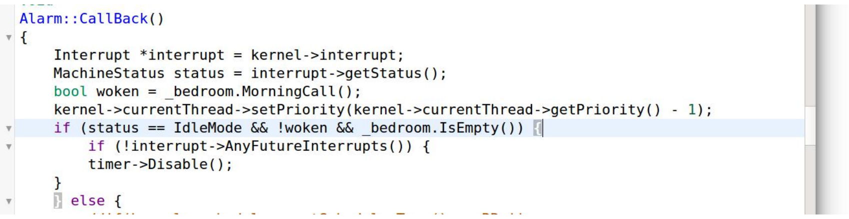
```cpp
        return;
    case SC_Sleep:
        val=kernel->machine->ReadRegister(4);
        cout << "Sleep time:" << val << "(ms)" << endl;
        kernel->alarm->WaitUntil(val);
        return;
```

（3）書寫 bedroom，全局計數



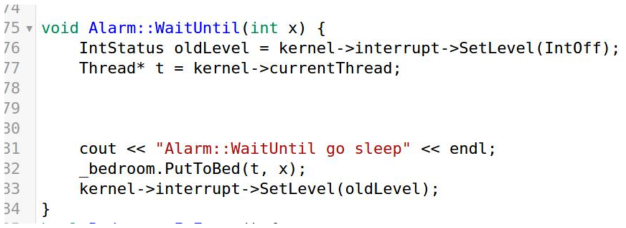
```cpp
class Bedroom {
    public:
        Bedroom():_current_interrupt(0) {};
        void PutToBed(Thread *t, int x);
    bool MorningCall();
    bool IsEmpty();
    private:
        class Bed {
            public:
                Bed(Thread* t, int x):
                    sleeper(t), when(x) {};
                Thread* sleeper;
                int when;
        };
```

（4）Callback 檢測何時喚醒



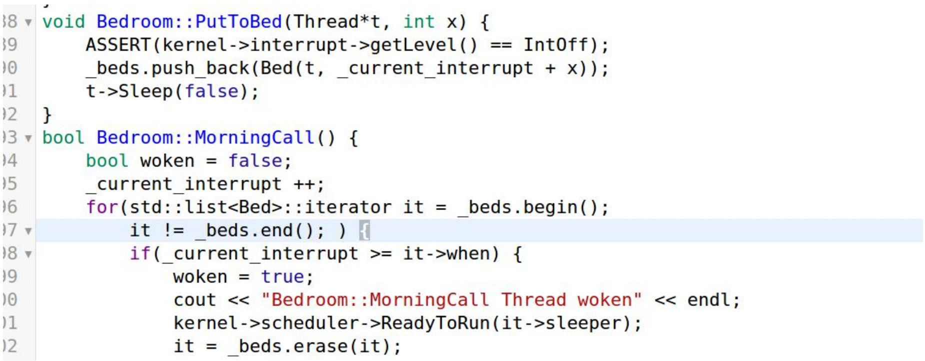
```
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool woken = _bedroom.MorningCall();
    kernel->currentThread->setPriority(kernel->currentThread->getPriority() - 1);
    if (status == IdleMode && !woken && _bedroom.IsEmpty()) {
        if (!interrupt->AnyFutureInterrupts()) {
        timer->Disable();
    }
    } else {
```

（5） WaitUntil()，將 thread 丟入 Bedroom 安眠。



```
void Alarm::WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;



    cout << "Alarm::WaitUntil go sleep" << endl;
    _bedroom.PutToBed(t, x);
    kernel->interrupt->SetLevel(oldLevel);
}
```

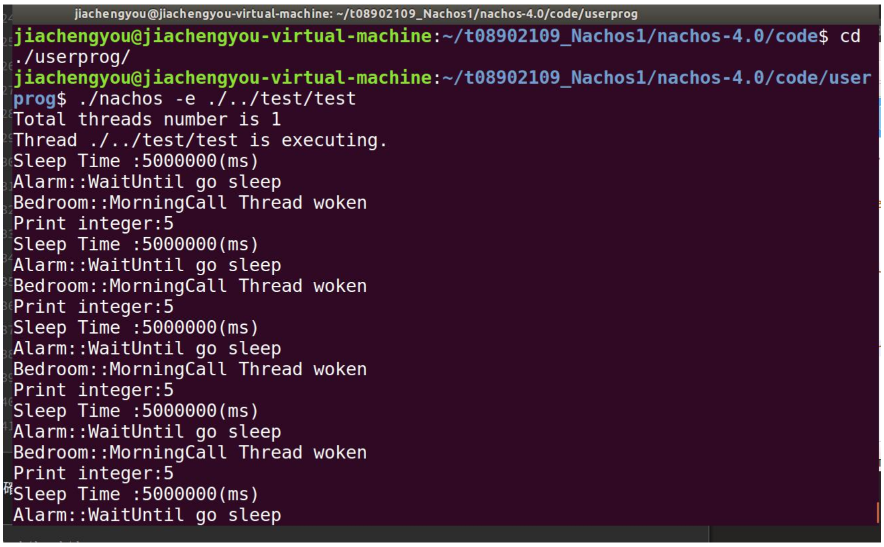
（6）bedroom 進入休眠，和喚醒休眠。



```
void Bedroom::PutToBed(Thread*t, int x) {
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    _beds.push_back(Bed(t, _current_interrupt + x));
    t->Sleep(false);
}
bool Bedroom::MorningCall() {
    bool woken = false;
    _current_interrupt ++;
    for(std::list<Bed>::iterator it = _beds.begin();
        it != _beds.end(); ) {
        if(_current_interrupt >= it->when) {
            woken = true;
            cout << "Bedroom::MorningCall Thread woken" << endl;
            kernel->scheduler->ReadyToRun(it->sleeper);
            it = _beds.erase(it);
```

3. Result

運行結果：



```
jiachengyou@jiachengyou-virtual-machine: ~/t08902109_Nachos1/nachos-4.0/code/userprog
jiachengyou@jiachengyou-virtual-machine:~/t08902109_Nachos1/nachos-4.0/code$ cd
./userprog/
jiachengyou@jiachengyou-virtual-machine:~/t08902109_Nachos1/nachos-4.0/code/user
prog$ ./nachos -e ./../test/test
Total threads number is 1
Thread ./../test/test is executing.
Sleep Time :5000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:5
Sleep Time :5000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:5
Sleep Time :5000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:5
Sleep Time :5000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:5
Sleep Time :5000000(ms)
Alarm::WaitUntil go sleep
```

二． CPU Scheduling

1. Motivation

目標：決定程式的執行次序。

有如下幾種方式：

FIFO (FCFS) 先來先服務

SJF 最短工作優先

Priority 最小優先權優先

RR (Round-robin)

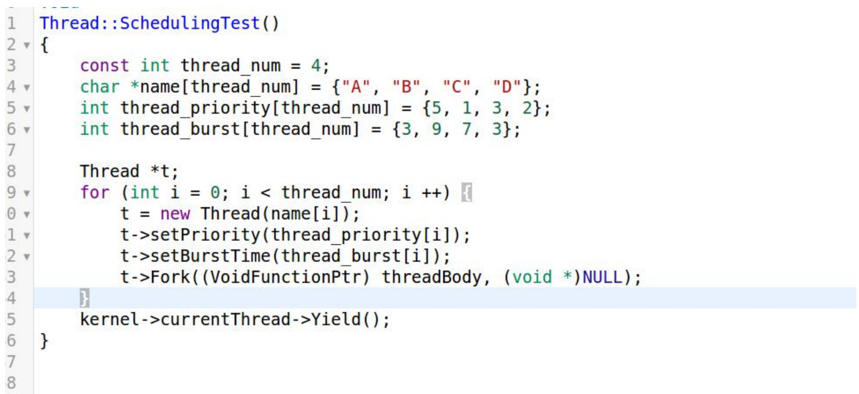
2. Implementation

```
cd code/threads
$ ./nachos FCFS
$ ./nachos SJF
$ ./nachos Priority
$ ./nachos RR
```

計劃：

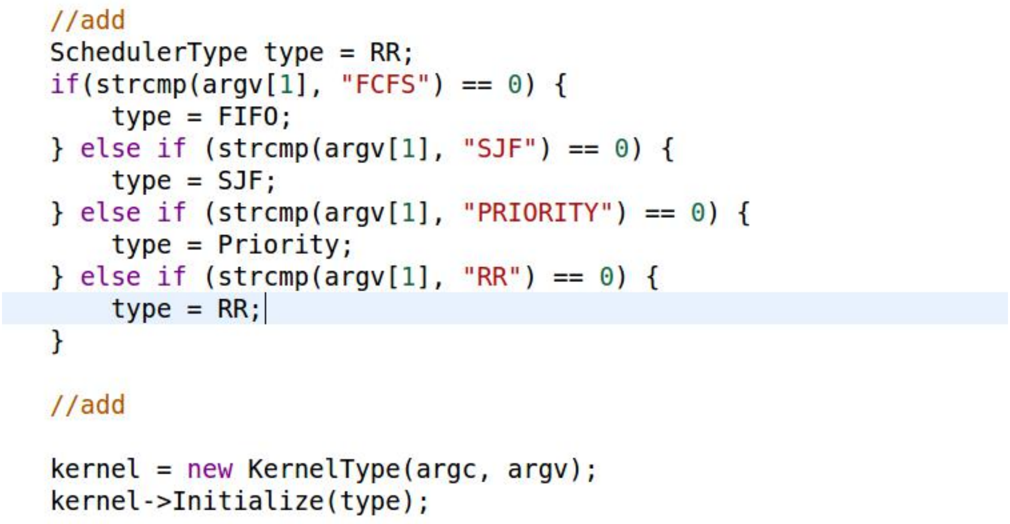
撰寫 self::test 函數，通過宣告不同的 compare function，實現不同類型的排程。

（1）書寫 test 測試



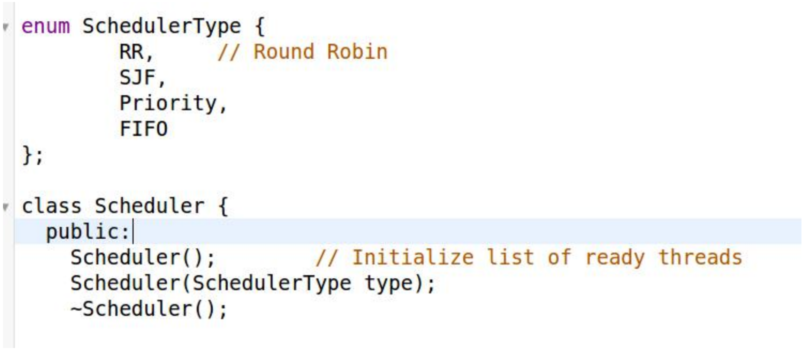
```
Thread::SchedulingTest()
{
    const int thread_num = 4;
    char *name[thread_num] = {"A", "B", "C", "D"};
    int thread_priority[thread_num] = {5, 1, 3, 2};
    int thread_burst[thread_num] = {3, 9, 7, 3};

    Thread *t;
    for (int i = 0; i < thread_num; i ++) {
        t = new Thread(name[i]);
        t->setPriority(thread_priority[i]);
        t->setBurstTime(thread_burst[i]);
        t->Fork((VoidFunctionPtr) threadBody, (void *)NULL);
    }
    kernel->currentThread->Yield();
}
```

（2）修改讀入參數

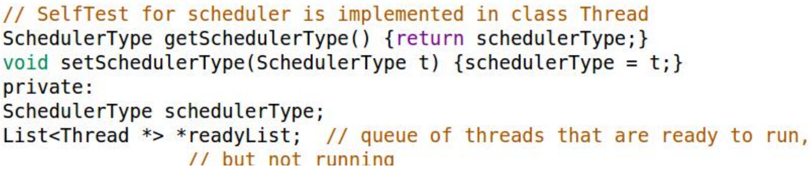


```
//add
SchedulerType type = RR;
if(strcmp(argv[1], "FCFS") == 0) {
    type = FIFO;
} else if (strcmp(argv[1], "SJF") == 0) {
    type = SJF;
} else if (strcmp(argv[1], "PRIORITY") == 0) {
    type = Priority;
} else if (strcmp(argv[1], "RR") == 0) {
    type = RR;
}

//add

kernel = new KernelType(argc, argv);
kernel->Initialize(type);
```

（3）定義 schedule type 和相關函數

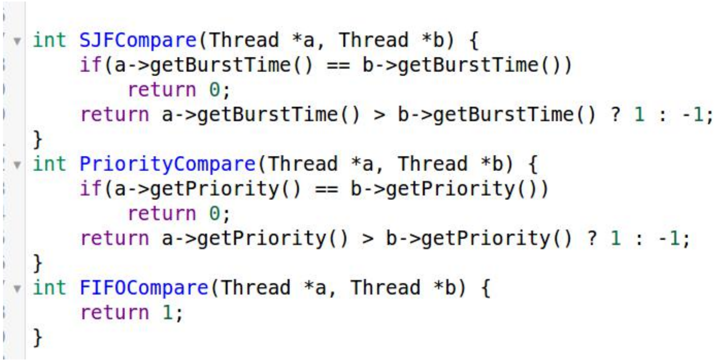


```
enum SchedulerType {
        RR,     // Round Robin
        SJF,
        Priority,
        FIFO
};

class Scheduler {
  public:
    Scheduler();        // Initialize list of ready threads
    Scheduler(SchedulerType type);
    ~Scheduler();
```

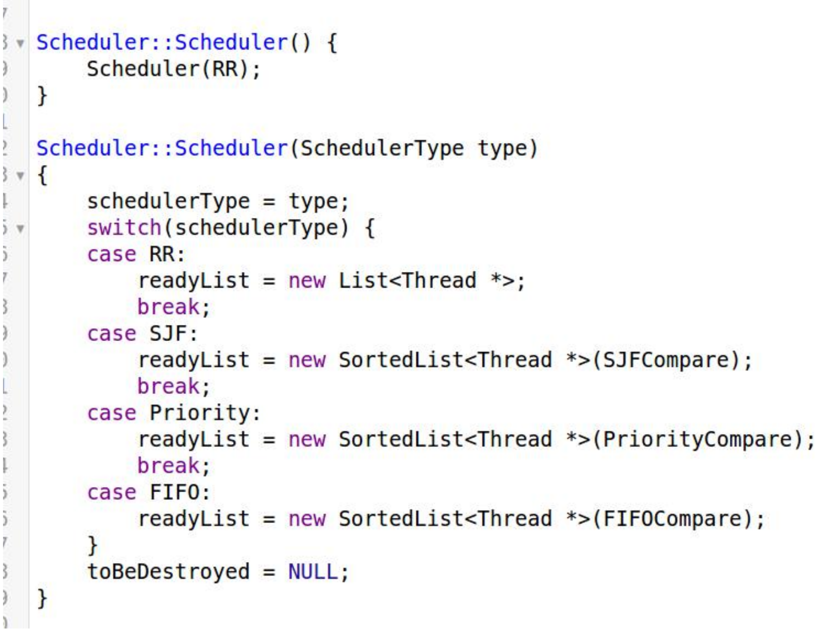


```
// SelfTest for scheduler is implemented in class Thread
SchedulerType getSchedulerType() {return schedulerType;}
void setSchedulerType(SchedulerType t) {schedulerType = t;}
private:
SchedulerType schedulerType;
List<Thread *> *readyList;  // queue of threads that are ready to run,
                // but not running
```

（4）不同類型的排程，宣告相對應的 compare function.

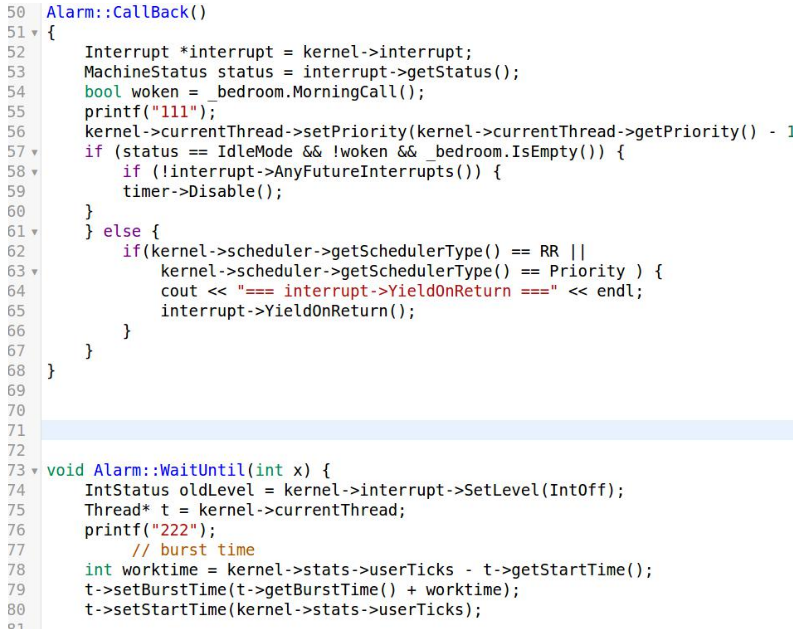


```
int SJFCompare(Thread *a, Thread *b) {
    if(a->getBurstTime() == b->getBurstTime())
        return 0;
    return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
}
int PriorityCompare(Thread *a, Thread *b) {
    if(a->getPriority() == b->getPriority())
        return 0;
    return a->getPriority() > b->getPriority() ? 1 : -1;
}
int FIFOCompare(Thread *a, Thread *b) {
    return 1;
}
```



```
Scheduler::Scheduler() {
    Scheduler(RR);
}

Scheduler::Scheduler(SchedulerType type)
{
    schedulerType = type;
    switch(schedulerType) {
    case RR:
        readyList = new List<Thread *>;
        break;
    case SJF:
        readyList = new SortedList<Thread *>(SJFCompare);
        break;
    case Priority:
        readyList = new SortedList<Thread *>(PriorityCompare);
        break;
    case FIFO:
        readyList = new SortedList<Thread *>(FIFOCompare);
    }
    toBeDestroyed = NULL;
}
```

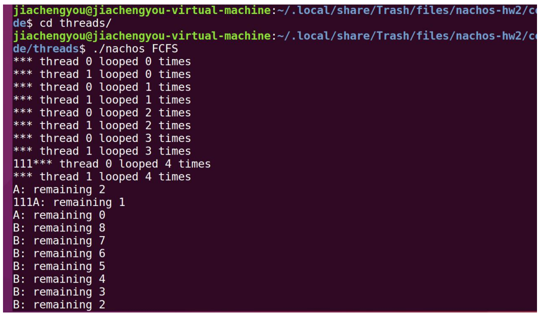
（5）修改到對應的 callback 和 waitUntil



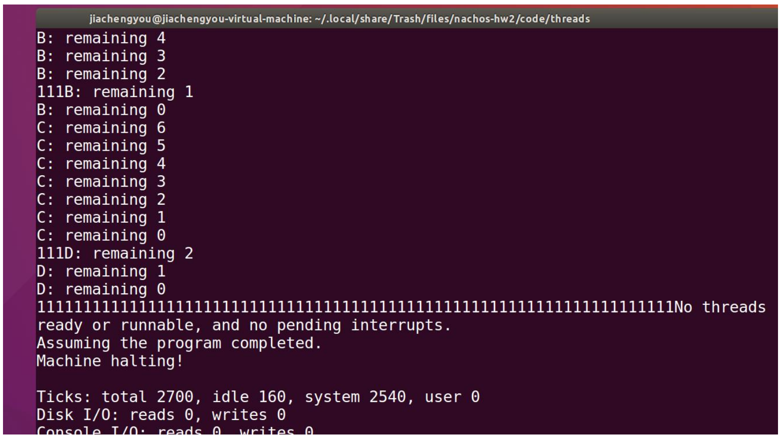
```
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool woken = _bedroom.MorningCall();
    printf("111");
    kernel->currentThread->setPriority(kernel->currentThread->getPriority() - 1
    if (status == IdleMode && !woken && _bedroom.IsEmpty()) {
        if (!interrupt->AnyFutureInterrupts()) {
        timer->Disable();
    }
    } else {
        if(kernel->scheduler->getSchedulerType() == RR ||
            kernel->scheduler->getSchedulerType() == Priority ) {
            cout << "=== interrupt->YieldOnReturn ===" << endl;
            interrupt->YieldOnReturn();
        }
    }
}



void Alarm::WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;
    printf("222");
         // burst time
    int worktime = kernel->stats->userTicks - t->getStartTime();
    t->setBurstTime(t->getBurstTime() + worktime);
    t->setStartTime(kernel->stats->userTicks);
```

3. Result

（1） (./threads) $ ./nachos FCFS



```
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/nachos-hw2/co
de$ cd threads/
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/nachos-hw2/co
de/threads$ ./nachos FCFS
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
111*** thread 0 looped 4 times
*** thread 1 looped 4 times
A: remaining 2
111A: remaining 1
A: remaining 0
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
```

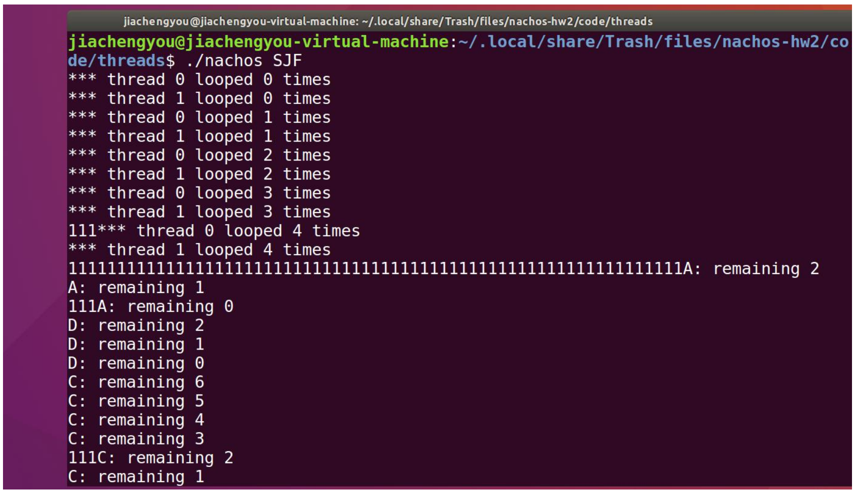


jiachengyou@jiachengyou-virtual-machine: ~/.local/share/Trash/files/nachos-hw2/code/threads

```
B: remaining 4
B: remaining 3
B: remaining 2
111B: remaining 1
B: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
111D: remaining 2
D: remaining 1
D: remaining 0
111111111111111111111111111111111111111111111111111111111111111111111No threads
ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2700, idle 160, system 2540, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
```

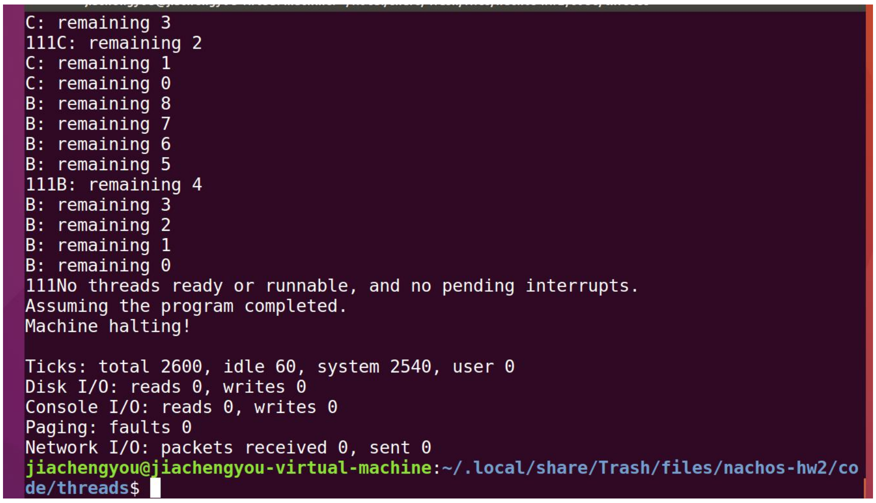
可以看出執行順序為 A,B,C,D，即先來先服務。

（2） (./threads) $ ./nachos SJF



jiachengyou@jiachengyou-virtual-machine: ~/.local/share/Trash/files/nachos-hw2/code/threads

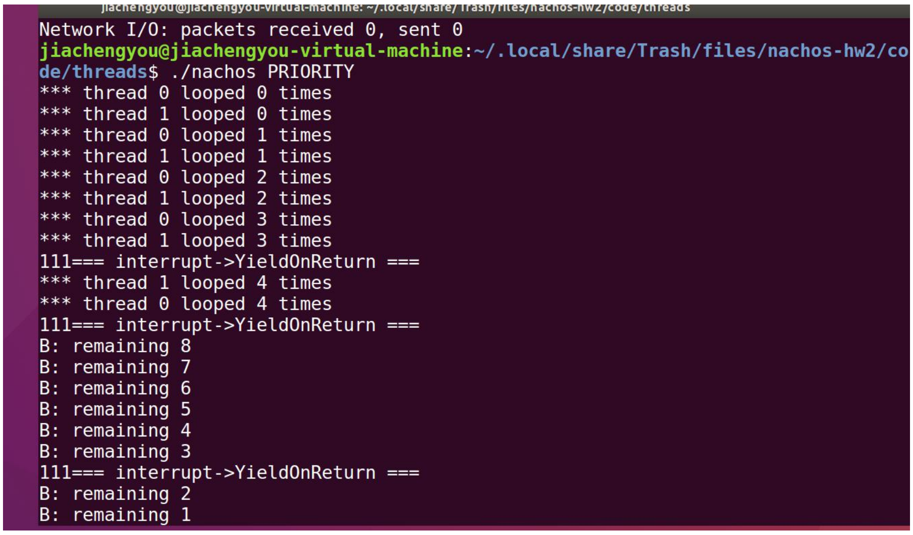
```
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/nachos-hw2/co
de/threads$ ./nachos SJF
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
111*** thread 0 looped 4 times
*** thread 1 looped 4 times
111111111111111111111111111111111111111111111111111111111111111A: remaining 2
A: remaining 1
111A: remaining 0
D: remaining 2
D: remaining 1
D: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
111C: remaining 2
C: remaining 1
```



```
C: remaining 3
111C: remaining 2
C: remaining 1
C: remaining 0
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
111B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
B: remaining 0
111No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2600, idle 60, system 2540, 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
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/nachos-hw2/co
de/threads$ 
```

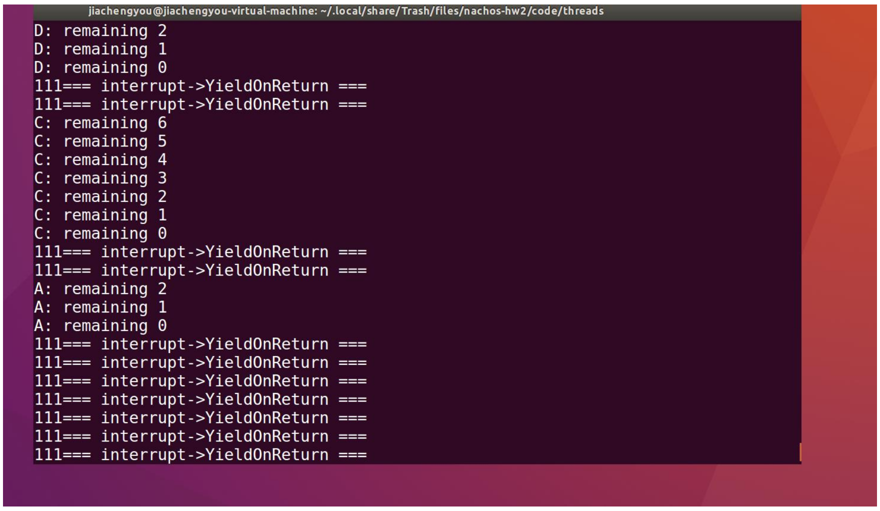
執行結果為 A,D,C,B，即最短工作優先，thread_burst 越小越先執行。

（3） (./threads) $ ./nachos　PRIORITY



jiachengyou@jiachengyou-virtual-machine: ~/.local/share/Trash/files/nachos-hw2/code/threads

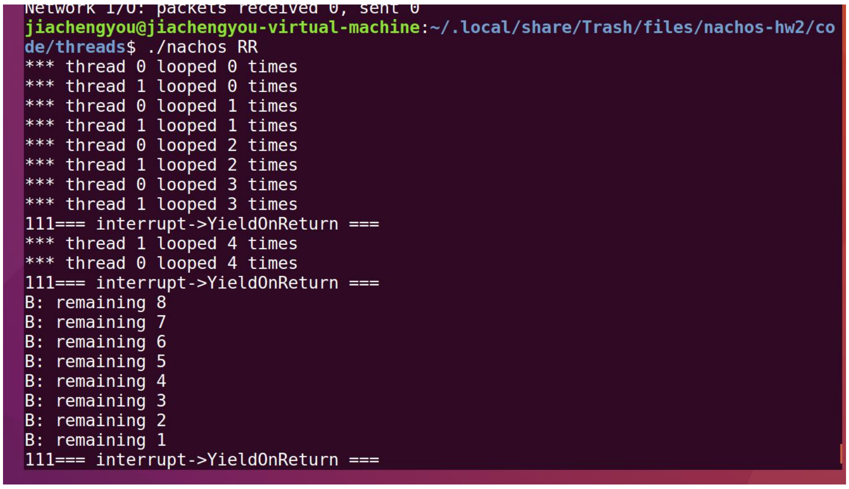
```
Network I/O: packets received 0, sent 0
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/nachos-hw2/co
de/threads$ ./nachos PRIORITY
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
111=== interrupt->YieldOnReturn ===
*** thread 1 looped 4 times
*** thread 0 looped 4 times
111=== interrupt->YieldOnReturn ===
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
111=== interrupt->YieldOnReturn ===
B: remaining 2
B: remaining 1
```



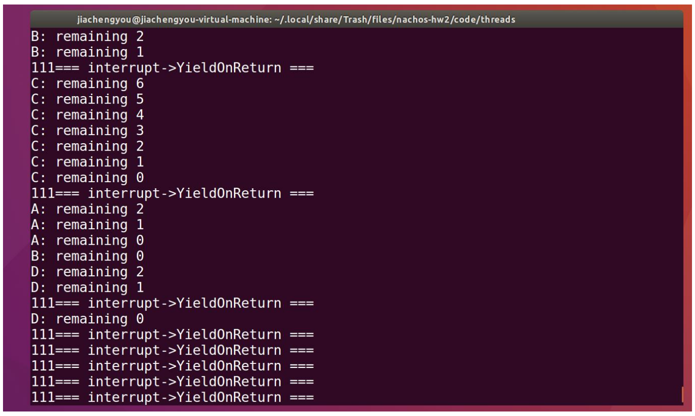
```
jiachengyou@jiachengyou-virtual-machine: ~/.local/share/Trash/files/nachos-hw2/code/threads
D: remaining 2
D: remaining 1
D: remaining 0
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
A: remaining 2
A: remaining 1
A: remaining 0
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
```

執行結果為 B,D,C,A，按照 priority 順序執行。

（4）(./threads) $ ./nachos RR



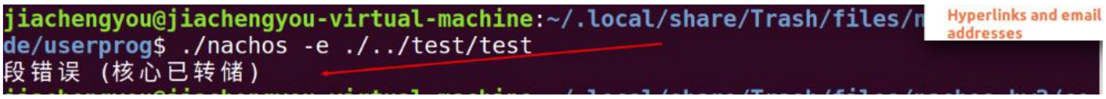
```
Network I/O: packets received 0, sent 0
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/nachos-hw2/co
de/threads$ ./nachos RR
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
111=== interrupt->YieldOnReturn ===
*** thread 1 looped 4 times
*** thread 0 looped 4 times
111=== interrupt->YieldOnReturn ===
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
111=== interrupt->YieldOnReturn ===
```



```
jiachengyou@jiachengyou-virtual-machine: ~/.local/share/Trash/files/nachos-hw2/code/threads
B: remaining 2
B: remaining 1
111=== interrupt->YieldOnReturn ===
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
111=== interrupt->YieldOnReturn ===
A: remaining 2
A: remaining 1
A: remaining 0
B: remaining 0
D: remaining 2
D: remaining 1
111=== interrupt->YieldOnReturn ===
D: remaining 0
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
111=== interrupt->YieldOnReturn ===
```

一些問題：

寫完 CPU schedule 時，如果執行原來的命令（如下），會出現段錯誤



```
jiachengyou@jiachengyou-virtual-machine:~/.local/share/Trash/files/n
de/userprog$ ./nachos -e ./../test/test
段错误 (核心已转储)
```

可能因為在寫第二個命令時修改到了不對的地方，因為時間關係，這個問題本次沒有解決。