Content

Reference

1. Nachos Project1 Report
https://github.com/taldehyde/os-project-1/blob/master/report.md
2. OS::NachOS::HW1
http://blog.terrynini.tw/tw/OS-NachOS-HW1/
3. 向 NachOS 4.0 作業進發 (2)
http://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/

I. Motivation

Motivation and the problem analysis

Part1: 撰寫自己的 Sleep function，將該 Thread 進入休眠。

Part2: 完成排程處理，在原始的 NachOS 4.0 中，基礎排程是 **Round-Robin**。
本次嘗試加入 FCFS 先來先服務、SJF 最短工作優先、測試不同排程處理

What's your plan to deal with the problem

Part1: 定義新的 system call code 並且宣告 **void Sleep(int time)**函數，並新增新的 exception 來處理 SC_Sleep system call

預計的工作如下：

1. 在/code/userprog/system.h 中加入 Sleep(int time)
2. 找一下 assembly language
3. 增加新的 exeption 來處理 SC_Sleep system call
4. 由於 kernel 存有 alarm，也就是說每隔固定一段間，就會呼叫 Alarm::CallBack()，因此，對於這個 alarm 來設計一個累加器 Bedroom::_current_interrupt 進行全局去記數，而 Alarm 的設定是每 100 ticks 就會產生一次 interrupt 並呼叫 Alarm::CallBack()
5. 當有程序呼叫 Sleep()時，會呼叫 WaitUntil()，然後將其丟入 Bedroom 安眠
6. 在 CallBlack()被呼叫時，去 Bedroom 檢查誰該醒來

Part2:

1. 在 Constructor 中決定要用哪一種類型的排程，並且宣告相對應的 compare function
2. 如果需要搶先設計，則在 Alarm::CallBack()決定是否要呼叫 interrupt->YieldOnReturn()查看是否有更需要優先的 process 要執行
3. 在 CPU Burst Time 的計算上，如果採用運行時估計，可以在利用執行 user program 的時間累加器進行計算
4. 分別在在每一個 kernel 宣告的地方(Initialized())都給一個決定 scheduler 類型的參數
5. 希望可以下這樣的 command 來執行

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

II. Implementation

Part1:

1. 定義 Sleep()函數

```
#define SC_ThreadYield  10
#define SC_PrintInt     11


void PrintInt(int number);      //my System Call

#endif /* IN_ASM */
```

Original Text

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

void PrintInt(int number);      //my System Call

void Sleep(int number);
#endif /* IN_ASM */
```

Changed Text

/code/userprog/syscall.h

2. 在 start.s 中修改 assembly language

.globl(c++ 寫的部分才能找得到) .ent Sleep(Sleep 的 entry 在這裡)

其餘部分大概跟其他 function 一樣，依樣畫葫蘆

```
        .end    PrintInt


/* dummy function to keep gcc happy */
        .globl  __main
        .ent    main
```

Original Text

```
        .end    PrintInt

        .globl  Sleep
        .ent    Sleep
Sleep:
        addiu   $2,$0,SC_Sleep
        syscall
        j       $31
        .end    Sleep

/* dummy function to keep gcc happy */
        .globl  __main
        .ent    main
```

Changed Text

/code/test/start.s

3. 修改 exception.cc, 收到 SC_Sleep 這個 syscall 編號時要做的行為

```
                    cout << "Print integer:" <<val << endl;
                    return;


/*              case SC_Exec:
```

Original Text

```
                    cout << "Print integer:" <<val << endl;
                    return;
                case SC_Sleep:
                    val=kernel->machine->ReadRegister(4);
                    kernel->alarm->WaitUntil(val);
                    return;
/*              case SC_Exec:
```

Changed Text

/code/userprog/exception.cc

4. 必須實作 kernel->alarm->WaitUntil(val); 設計累加器 Bedroom::_current_interrupt。當有程序呼叫 Sleep() 時，會呼叫 WaitUntil()，然後將其丟入 Bedroom 安眠。

```
#include "timer.h"


// The following class defines a software alarm clock.
class Alarm : public CallBackObj {
  public:
    Alarm(bool doRandomYield);  // Initialize the timer, and callback
                                // to "toCall" every time slice.
    ~Alarm() { delete timer; }

    void WaitUntil(int x);      // suspend execution until time > now + x

  private:
    Timer *timer;               // the hardware timer device

    void CallBack();            // called when the hardware
```

Original Text

```
#include "timer.h"
#include "thread.h"
#include <list>

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;
        };

    int _current_interrupt;
    std::list<Bed> _beds;
};

// The following class defines a software alarm clock.
class Alarm : public CallBackObj {
  public:
    Alarm(bool doRandomYield);  // Initialize the timer, and callback
                                // to "toCall" every time slice.
    ~Alarm() { delete timer; }

    void WaitUntil(int x);      // suspend execution until time > now + x

  private:
    Timer *timer;               // the hardware timer device
    Bedroom _bedroom;
    void CallBack();            // called when the hardware
```

Changed Text

/code/threads/alarm.h

5. 在 CallBlack()被呼叫時，來去 Bedroom 檢查誰該醒來。WaitUntil 的定義

```
53    MachineStatus status = interrupt->getStatus();
54
55    if (status == IdleMode) {      // is it time to quit?
56        if (!interrupt->AnyFutureInterrupts()) {
57            timer->Disable();    // turn off the timer
58        }
59    } else {                    // there's someone to preempt
60        interrupt->YieldOnReturn();
61    }
62 }
```

Original Text

```
53    MachineStatus status = interrupt->getStatus();
54    bool woken = _bedroom.MorningCall();
55
56    if (status == IdleMode && !woken && _bedroom.IsEmpty())
      {      // is it time to quit?
57        if (!interrupt->AnyFutureInterrupts()) {
58            timer->Disable();    // turn off the timer
59        }
60    }
61    else {                      // there's someone to preempt
62        //interrupt->YieldOnReturn();
63        if(kernel->scheduler->getSchedulerType() == RR ||
64            kernel->scheduler->getSchedulerType() == Priority ) {
65    //          interrupt->YieldOnReturn();
66            interrupt->YieldOnReturn();
67        }
68    }
69 }
70
71 void Alarm::WaitUntil(int x) {
72    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
73    Thread* t = kernel->currentThread;
74     // burst time
75    int worktime = kernel->stats->userTicks - t->getStartTime();
76    t->setBurstTime(t->getBurstTime() + worktime);
77    t->setStartTime(kernel->stats->userTicks);
78    cout << "Alarm::WaitUntil go sleep" << endl;
79    _bedroom.PutToBed(t, x);
80    kernel->interrupt->SetLevel(oldLevel);
81 }
82
83 bool Bedroom::IsEmpty() {
84    return _beds.size() == 0;
85 }
86
87 void Bedroom::PutToBed(Thread*t, int x) {
88    ASSERT(kernel->interrupt->getLevel() == IntOff);
89    _beds.push_back(Bed(t, _current_interrupt + x));
90    t->Sleep(false);
91 }
92
93 bool Bedroom::MorningCall() {
94    bool woken = false;
95    _current_interrupt ++;
96    for(std::list<Bed>::iterator it = _beds.begin();
97        it != _beds.end(); ) {
98        if(_current_interrupt >= it->when) {
99            woken = true;
100           cout << "Bedroom::MorningCall Thread woken" << endl;
101           kernel->scheduler->ReadyToRun(it->sleeper);
102           it = _beds.erase(it);
103       } else {
104           it++;
105       }
106   }
107   return woken;
108 }
```

Changed Text

/code/threads/alarm.cc

Part2:

1. 依照助教投影片指示先設計 main.cc，接著設計標頭檔

```
76    DEBUG(dbgThread, "Entering main");
77    kernel = new KernelType(argc, argv);
78    kernel->Initialize();
```

Original Text

```
76    DEBUG(dbgThread, "Entering main");
77  //
78    SchedulerType type = RR;
79    if(strcmp(argv[1], "FCFS") == 0) {
80        type = FIFO;
81    } else if (strcmp(argv[1], "SJF") == 0) {
82        type = SJF;
83    } else if (strcmp(argv[1], "PRIORITY") == 0) {
84        type = Priority;
85    } else if (strcmp(argv[1], "RR") == 0) {
86        type = RR;
87    }
88  //
89    kernel = new KernelType(argc, argv);
90    kernel->Initialize(type);
```

Changed Text

/code/threads/main.cc

2. 為了實作多種不同的 scheduler，先定義 enum function 並讓程式可以藉由 [- SchedulerType]來切換排程模式，在 SchedulerType 中多增加 FIFO

```
20 enum SchedulerType {
21         RR,     // Round Robin
22         SJF,
23         Priority
24 };
25
26 class Scheduler {
27   public:
28         Scheduler();            // Initialize list of ready threads
29         ~Scheduler();                           // De-allocate ready list
30
31         void ReadyToRun(Thread* thread);
32                                 // Thread can be dispatched.
33         Thread* FindNextToRun();        // Dequeue first thread on the ready
34                                 // list, if any, and return thread.
35         void Run(Thread* nextThread, bool finishing);
36                                 // Cause nextThread to start running
37         void CheckToBeDestroyed();      // Check if thread that had been
38                                 // running needs to be deleted
39         void Print();                   // Print contents of ready list
40
41     // SelfTest for scheduler is implemented in class Thread
42
43   private:
```

Original Text

```
20 enum SchedulerType {
21         RR,     // Round Robin
22         SJF,
23         Priority,
24         FIFO
25 };
26
27 class Scheduler {
28   public:
29         Scheduler();
30         Scheduler(SchedulerType type);          //
   Initialize list of ready threads
31         ~Scheduler();                           // De-allocate ready list
32
33         void ReadyToRun(Thread* thread);
34                                 // Thread can be dispatched.
35         Thread* FindNextToRun();        // Dequeue first thread on the ready
36                                 // list, if any, and return thread.
37         void Run(Thread* nextThread, bool finishing);
38                                 // Cause nextThread to start running
39         void CheckToBeDestroyed();      // Check if thread that had been
40                                 // running needs to be deleted
41         void Print();                   // Print contents of ready list
42
43     // SelfTest for scheduler is implemented in class Thread
44
45         SchedulerType getSchedulerType() {return schedulerType;}
46         void setSchedulerType(SchedulerType t) {schedulerType = t;}
47   private:
```

Changed Text

/code/threads/scheduler.h

3. 在 Constructor 中決定要用哪一種類型的排程，並且宣告相對應的 compare function

```
32 Scheduler::Scheduler()
33 {
34 //      schedulerType = type;
35         readyList = new List<Thread *>;
36         toBeDestroyed = NULL;
```

Original Text

```
32 int SJFCompare(Thread *a, Thread *b) {
33     if(a->getBurstTime() == b->getBurstTime())
34         return 0;
35     return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
36 }
37
38 int PriorityCompare(Thread *a, Thread *b) {
39     if(a->getPriority() == b->getPriority())
40         return 0;
41     return a->getPriority() > b->getPriority() ? 1 : -1;
42 }
43
44 int FIFOCompare(Thread *a, Thread *b) {
45     return 1;
46 }
47
48 Scheduler::Scheduler()
49 {
50 //      schedulerType = type;
51         readyList = new List<Thread *>;
52         toBeDestroyed = NULL;
53 }
54 Scheduler::Scheduler(SchedulerType type)
55 {
56     schedulerType = type;
57     switch(schedulerType) {
58     case RR:
59         readyList = new List<Thread *>;
60         break;
61     case SJF:
62         readyList = new SortedList<Thread *>(SJFCompare);
63         break;
64     case Priority:
65         readyList = new SortedList<Thread *>(PriorityCompare);
66         break;
67     case FCFS:
68         readyList = new SortedList<Thread *>(FIFOCompare);
69     }
70     toBeDestroyed = NULL;
```

Changed Text

/code/threads/scheduler.cc

4. 在 Alarm::CallBack()決定是否要呼叫 interrupt->YieldOnReturn()查看是否有更需要優先的 process 要執行

```
49 void
50 Alarm::CallBack()
51 {
52     Interrupt *interrupt = kernel->interrupt;
53     MachineStatus status = interrupt->getStatus();
54
55     if (status == IdleMode) {      // is it time to quit?
56         if (!interrupt->AnyFutureInterrupts()) {
57             timer->Disable();    // turn off the timer
58         }
59     } else {                    // there's someone to preempt
60         interrupt->YieldOnReturn();
61     }
```

```
49 void
50 Alarm::CallBack()
51 {
52     Interrupt *interrupt = kernel->interrupt;
53     MachineStatus status = interrupt->getStatus();
54     bool woken = _bedroom.MorningCall();
55
56     if (status == IdleMode && !woken && _bedroom.IsEmpty()) {      // is it time to quit?
57         if (!interrupt->AnyFutureInterrupts()) {
58             timer->Disable();    // turn off the timer
59         }
60     }
61     else {                      // there's someone to preempt
62         //interrupt->YieldOnReturn();
63         if(kernel->scheduler->getSchedulerType() == RR ||
64             kernel->scheduler->getSchedulerType() == Priority ) {
65     //          interrupt->YieldOnReturn();
66             interrupt->YieldOnReturn();
67         }
68     }
69 }
```

kernel->stats->userTicks 是執行 user program 的時間累加器

```
71 void Alarm::WaitUntil(int x) {
72     IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
73     Thread* t = kernel->currentThread;
74      // burst time
75     int worktime = kernel->stats->userTicks - t->getStartTime();
76     t->setBurstTime(t->getBurstTime() + worktime);
77     t->setStartTime(kernel->stats->userTicks);
78     cout << "Alarm::WaitUntil go sleep" << endl;
79     _bedroom.PutToBed(t, x);
80     kernel->interrupt->SetLevel(oldLevel);
81 }
82
83 bool Bedroom::IsEmpty() {
84     return _beds.size() == 0;
85 }
86
87 void Bedroom::PutToBed(Thread*t, int x) {
88     ASSERT(kernel->interrupt->getLevel() == IntOff);
89     _beds.push_back(Bed(t, _current_interrupt + x));
90     t->Sleep(false);
91 }
92
93 bool Bedroom::MorningCall() {
94     bool woken = false;
95     _current_interrupt ++;
96     for(std::list<Bed>::iterator it = _beds.begin();
97         it != _beds.end(); ) {
98         if(_current_interrupt >= it->when) {
99             woken = true;
100            cout << "Bedroom::MorningCall Thread woken" << endl;
101            kernel->scheduler->ReadyToRun(it->sleeper);
102            it = _beds.erase(it);
103        } else {
104            it++;
105        }
106    }
107    return woken;
```

Original Text Changed Text

/code/threads/alarm.cc

5. 為了測試排程的結果，多增加一個 Thread::SchedulingTest()的函式，

Name	A	B	C	D
Priority	5	1	3	2
Burst Time	3	9	7	3

並設置各程式的 burst time 和 priority 特性，各個程式特性分別設置如上表，

```
436     SimpleThread(0);
437 }
```

```
438     SimpleThread(0);
439 }
440 // ----------------------------------------------------------------------
441 // Test schedular
442 // ----------------------------------------------------------------------
443 void
444 threadBody() {
445         Thread *thread = kernel->currentThread;
446         while (thread->getBurstTime() > 0) {
447                 thread->setBurstTime(thread->getBurstTime() - 1);
448                 kernel->interrupt->OneTick();
449                 printf("%s: remaining %d\n", kernel->currentThread->getName(), kernel->cu
rentThread->getBurstTime());
450         }
451 }
452 void
453 Thread::SchedulingTest()
454 {
455         const int thread_num = 4;
456         char *name[thread_num] = {"A", "B", "C", "D"};
457         int thread_priority[thread_num] = {5, 1, 3, 2};
458         int thread_burst[thread_num] = {3, 9, 7, 3};
459
460
461         Thread *t;
462         for (int i = 0; i < thread_num; i ++) {
463                 t = new Thread(name[i]);
464                 t->setPriority(thread_priority[i]);
465                 t->setBurstTime(thread_burst[i]);
466                 t->Fork((VoidFunctionPtr) threadBody, (void *)NULL);
467         }
468         kernel->currentThread->Yield();
469 }
```

Original Text Changed Text

/code/threads/thread.cc

6. 必須在 class Thread 增加 setPriority(), setBurstTime(), SchedulingTest() … 等 method header。

```
109
110
111
110  private:
111    // some of the private data for this class is listed above
```

```
109
110
111    void setBurstTime(int t)    {burstTime = t;}
112    int getBurstTime()          {return burstTime;}
113    void setStartTime(int t)    {startTime = t;}
114    int getStartTime()          {return startTime;}
115    void setPriority(int t)     {execPriority = t;}
116    int getPriority()           {return execPriority;}
117    static void SchedulingTest();
118  private:
119    // some of the private data for this class is listed above
120
121    // morris add
122    int burstTime;      // predicted burst time
123    int startTime;      // the start time of the thread
124    int execPriority;   // the execute priority of the thread
```

Original Text　　Changed Text

/code/threads/thread.h

7. 將 testcode 加入到 ThreadedKernel 中。

```
46 //----------------------------------------------------------------------
47
48 void
49 ThreadedKernel::Initialize()
50 {
51     stats = new Statistics();           // collect statistics
52     interrupt = new Interrupt;          // start up interrupt handling
53     scheduler = new Scheduler();    // initialize the ready queue
111    currentThread->SelfTest();   // test thread switching
112
113                                // test semaphore operation
114    semaphore = new Semaphore("test", 0);
```

```
46 //----------------------------------------------------------------------
47 void
48 ThreadedKernel::Initialize() {
49     Initialize(RR);
50 }
51
52 void
53 ThreadedKernel::Initialize(SchedulerType type)
54 {
55     stats = new Statistics();           // collect statistics
56     interrupt = new Interrupt;          // start up interrupt handling
57     scheduler = new Scheduler(type);          // initialize the ready queue
115    currentThread->SelfTest();   // test thread switching
116    Thread::SchedulingTest();
117                                // test semaphore operation
118    semaphore = new Semaphore("test", 0);
```

Original Text　　Changed Text

/code/threads/ kernel.cc

8. 後來發現 make 有錯誤，經過上網一查 ref[3] 向 NachOS 4.0 作業進發 (2) https://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/問題是發生於我們在 /threads 下修改 main.cc 的關係，所以必須在每一個 kernel 宣告地方都宣告的地方(Initialized())都給一個決定 scheduler 類型的參數。

```
#include "synchdisk.h"

class SynchDisk;
class UserProgKernel : public ThreadedKernel {
  public:
    UserProgKernel(int argc, char **argv);
                                // Interpret command line arguments
    ~UserProgKernel();          // deallocate the kernel

    void Initialize();          // initialize the kernel

```

Original Text

```
#include "synchdisk.h"
#include "../threads/scheduler.h"

class SynchDisk;
class UserProgKernel : public ThreadedKernel {
  public:
    UserProgKernel(int argc, char **argv);
                                // Interpret command line arguments
    ~UserProgKernel();          // deallocate the kernel

    void Initialize();          // initialize the kernel
    void Initialize(SchedulerType type);

```

Changed Text

userprog/userkernel.h

```
// UserProgKernel::Initialize
//      Initialize Nachos global data structures.
//----------------------------------------------------------------------

void
UserProgKernel::Initialize()
{
    ThreadedKernel::Initialize();   // init multithreading



    machine = new Machine(debugUserProg);
```

Original Text

```
// UserProgKernel::Initialize
//      Initialize Nachos global data structures.
//----------------------------------------------------------------------

void
UserProgKernel::Initialize()
{
    Initialize(RR);
}
void
UserProgKernel::Initialize(SchedulerType type)
{
    ThreadedKernel::Initialize(type);   // init multithreading

    machine = new Machine(debugUserProg);
```

Changed Text

userprog/userkernel.cc

```
#include "userkernel.h"

class PostOfficeInput;
class PostOfficeOutput;

class NetKernel : public UserProgKernel {
  public:
    NetKernel(int argc, char **argv);
                                // Interpret command line arguments
    ~NetKernel();               // deallocate the kernel

    void Initialize();          // initialize the kernel
```

Original Text

```
#include "userkernel.h"
#include "../threads/scheduler.h"

class PostOfficeInput;
class PostOfficeOutput;

class NetKernel : public UserProgKernel {
  public:
    NetKernel(int argc, char **argv);
                                // Interpret command line arguments
    ~NetKernel();               // deallocate the kernel

    void Initialize();          // initialize the kernel
    void Initialize(SchedulerType);
```

Changed Text

network/netkernel.h

```
//----------------------------------------------------------------------

void
NetKernel::Initialize()
{
    UserProgKernel::Initialize();   // init other kernel data structs

    postOfficeIn = new PostOfficeInput(10);
    postOfficeOut = new PostOfficeOutput(reliability, 10);
```

Original Text

```
//----------------------------------------------------------------------
void
NetKernel::Initialize() {
        Initialize(RR);
}
void
NetKernel::Initialize(SchedulerType type)
{
    UserProgKernel::Initialize(type);       // init other kernel data structs

    postOfficeIn = new PostOfficeInput(10);
    postOfficeOut = new PostOfficeOutput(reliability, 10);
```

Changed Text

network/netkernel.cc

III. Result

Experiment result and some discussion

Part1:

1. 新增兩個有用到 sleep 的程式，檢查正確性，
   sleep2 的週期是 sleep 的 1/10

```c
#include "syscall.h"
main() {
  int n;
  for (n = 0; n < 5; n++) {
    PrintInt(1111);
    Sleep(1000000);
  }
}
```

sleep1.c

```c
#include "syscall.h"
main() {
  int n;
  for (n = 0; n < 5; n++) {
    PrintInt(2222);
    Sleep(100000);
  }
}
```

sleep2.c

2. 修改 Makefile 加入編譯，把 coff 轉成 NachOS 自己定義的 noff 格式

```
all: halt shell matmult sort test1 test2

start.o: start.s ../userprog/syscall.h
```

```
all: halt shell matmult sort test1 test2 sleep1 sleep2

start.o: start.s ../userprog/syscall.h

test: test.o start.o
	$(LD) $(LDFLAGS) start.o test.o -o test.coff
	../bin/coff2noff test.coff test

sleep1: sleep1.o start.o
	$(LD) $(LDFLAGS) start.o sleep1.o -o sleep1.coff
	../bin/coff2noff sleep1.coff sleep1

sleep2: sleep2.o start.o
	$(LD) $(LDFLAGS) start.o sleep2.o -o sleep2.coff
	../bin/coff2noff sleep2.coff sleep2
```

/test/Makefile

3. 下 make 指令產出執行檔
4. 執行 ./nachos -e ../test/sleep1 -e ../test/sleep2

前兩次會參雜 10 次的 PrintInt(1111)
但會進入休眠給 sleep2
sleep2 會趁 sleep1 還在休眠時
執行完時，因為我只有 iteration 五次所以還
會再等一下再把 PrintInt(2222)執行完五次

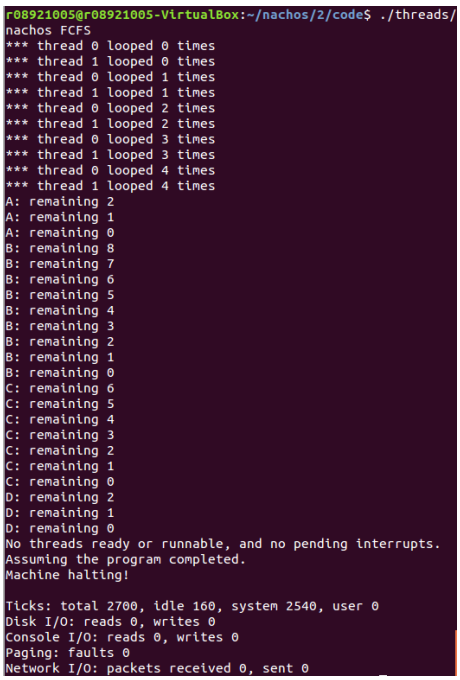
```
r08921005@r08921005-VirtualBox:~/nachos/2/code/userprog$
ep1 -e ../test/sleep2
Total threads number is 2
Thread ../test/sleep1 is executing.
Thread ../test/sleep2 is executing.
Print integer:1111
Sleep Time 1000000(ms)
Alarm::WaitUntil go sleep
Print integer:2222
Sleep Time 100000(ms)
Alarm::WaitUntil go sleep
Print integer:2222
Sleep Time 100000(ms)
Alarm::WaitUntil go sleep
Print integer:2222
Sleep Time 100000(ms)
Alarm::WaitUntil go sleep
Print integer:2222
Sleep Time 100000(ms)
Alarm::WaitUntil go sleep
Print integer:2222
Sleep Time 100000(ms)
Alarm::WaitUntil go sleep
return value:0
Print integer:1111
Sleep Time 1000000(ms)
Alarm::WaitUntil go sleep
Print integer:1111
Sleep Time 1000000(ms)
Alarm::WaitUntil go sleep
Print integer:1111
Sleep Time 1000000(ms)
Alarm::WaitUntil go sleep
Print integer:1111
Sleep Time 1000000(ms)
Alarm::WaitUntil go sleep
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 500000100, idle 499999560, system 260, user
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
r08921005@r08921005-VirtualBox:~/nachos/2/code/userprog$
```

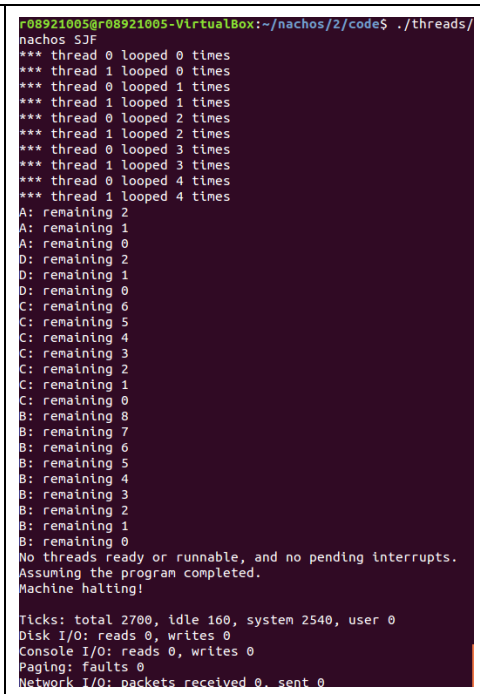
Part2:

對應剛剛 Thread::SchedulingTest()的函式，各個程式特性分別設置如下表，

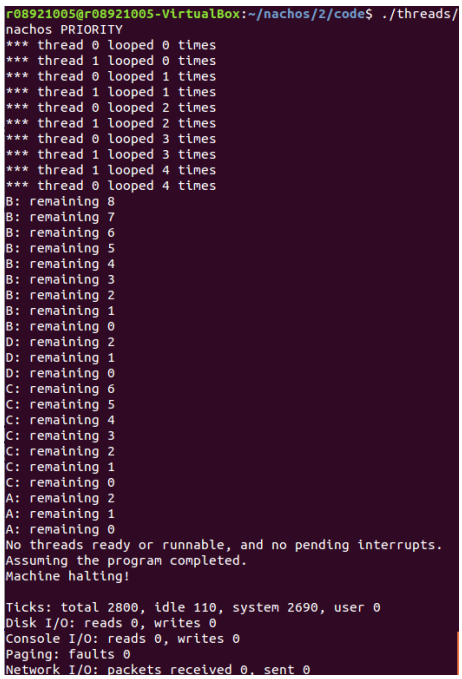
Name	A	B	C	D
Priority	5	1	3	2
Burst Time	3	9	7	3



./threads/nachos FCFS



./threads/nachos SJF



./threads/nachos PRIORITY

分別測試 FCFS、SJF、Priority，討論其結果

FCFS	確實是按照先來後到的順序，也就是 ABCD 的順序做執行
SJF	具備最長時間 CPU-Burst Time 的 B 程序確實是最後一個執行的， 然而同時是最短長度的兩個程序 A 與 D 是最先被執行的
Priority	確實是優先等地的順序，也就是 BDCA(1235)的順序做執行