# OS2022 Project2

> Name : 郭忠翔
>
> Department : 機械所控制組
>
> ID : R10522845

## Motivation

### Motivation and the problem analysis

- System Call : 撰寫Sleep function，將Thread進入Sleep mode。
- CPU scheduling : 在Nachos4.0中的默認排程是Round-robin(RR)，要實作多種排程，要在`code/threads/main.cc`中的`initialize`給參數輸入來選定排程

```
// code/threads/main.cc
int
main(int argc, char **argv)
{
    ...
    kernel = new KernelType(argc, argv);
    kernel->Initialize();                //need to pass parameter
    ...
}
```

### What your plan to deal with the problem

- System Call : 依照作業中的comments，先寫測試程式`sleep.c`，再去`code/test`中的`Makefile`做修改，實作出sleep function再去執行測試程式
- CPU scheduling : 首先先寫自己的`SelfTest()`，測試寫出的排程是否正確，修改`main.cc`中initialize的問題

## Implementation

### How do you implement to solve the problem in NachOS

- System Call

  先寫測試程式`sleep.c`

```
#include "syscall.h"
main() {
    int i;
    for(i = 0; i < 3; i++) {
        Sleep(10000000);
        PrintInt(i);
    }
    return 0;
}
```

去`Makefile`做修改

```
// code/test/Makefile
...
all: halt shell matmult sort test1 test2 sleep
...
sleep: sleep.o start.o
    $(LD) $(LDFLAGS) start.o sleep.o -o sleep.coff  // use tab
rather than space, or error will occur
    ../bin/coff2noff sleep.coff sleep
```

開始實作sleep function，先宣告function

```
// code/usrprog/syscall.h
...
#define SC_Sleep    12
...
void Sleep(int number);
```

Prepare register for Sleep()

```
// code/test/start.s
...
PrintInt:
    addiu   <span>$2,$</span><!-- Has MathJax -->0,SC_PrintInt
    syscall
    j       $31
    .end    PrintInt
    .globl  Sleep
    .ent    Sleep
Sleep:
    addiu   $2,$0,SC_Sleep
    syscall
    j   $31
    .end    Sleep
...
```

在`exception.cc`中加入sleep的case並使用`WaitUntil`讓系統可以處理

```
// code/userprog/exception.cc
...
case SC_Exit:
    DEBUG(dbgAddr, "Program exit\n");
    val=kernel->machine->ReadRegister(4);
    cout << "return value:" << val << endl;
    kernel->currentThread->Finish();
        break;
case SC_Sleep:
    val=kernel->machine->ReadRegister(4);
    cout << "Sleep Time " << val << "(ms) " << endl;
    kernel->alarm->WaitUntil(val);
    return;
```

在`alarm.h`中，`WaitUntil`雖然有宣告，但是在`alarm.cc`卻沒有實作出來，只有`CallBack`，因此運作的時候可能會有很多的thread去睡覺，所以要有一個List儲存這些睡眠中的thread，也要設定一個類似鬧鐘的計數器`Bedroom::_current_interrupt`儲存他們各自的起床時間

```
// code/threads/alarm.h
...
#include <list>
#include "thread.h"
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;
};
class Alarm : public CallBackObj {
    ...
    private:
    Bedroom _bedroom;
    ...
```

實作method

```
// code/threads/alarm.cc
void Alarm::CallBack() {
    ...
    bool woken = _bedroom.MorningCall();
}
void Alarm::WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;

    int worktime = kernel->stats->userTicks - t-
>getStartTime();
    t->setBurstTime(t->getBurstTime() + worktime);
    t->setStartTime(kernel->stats->userTicks);
    cout << "Alarm::WaitUntil go sleep" << endl;
    _bedroom.PutToBed(t, x);
    kernel->interrupt->SetLevel(oldLevel);
}
bool Bedroom::IsEmpty() {
    return _beds.size() == 0;
}
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);
            } else {
                it++;
            }
        }
        return woken;
}
```

- CPU Scheduling

  先修改`main.cc`，讓initialize可輸入參數

```
// code/threads/main.cc
int
main(int argc, char **argv)
{
    ...
    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;
    }
    ...
    kernel = new KernelType(argc, argv);
    kernel->Initialize(type);               //pass parameter
    ...
}
```

  因為有修改`Initialize`，所以必須在每一個 kernel 宣告地方都給一個決定`SchedulerType`的參數，因為在不同的地方但只要加上宣告，因此我寫在同一格

```
#include "../threads/scheduler.h"
// code/userprog/userkernel.h
class UserProgKernel : public ThreadedKernel {
  public:
    ...
    void Initialize();      // initialize the kernel
    void Initialize(SchedulerType type);
    ...
```

```
// code/network/netkernel.h
class NetKernel : public UserProgKernel {
  public:
    ...
    void Initialize();      // initialize the kernel
    void Initialize(SchedulerType);
    ...
// code/threads/kernel.h
class ThreadedKernel {
  public:
    ...
    void Initialize(SchedulerType type);
    void Initialize();          // initialize the kernel --
separated
    ...
```

實作`Initialize(SchedulerType type)`

```
// code/userprog/userkernel.cc
void
UserProgKernel::Initialize()
{
    Initialize(RR);
}
void
UserProgKernel::Initialize(SchedulerType type)
{
    ThreadedKernel::Initialize(type);   // init multithreading
    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}
// code/network/netkernel.cc
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);
}
// code/threads/kernel.cc
void
ThreadedKernel::Initialize(SchedulerType type)
{
    stats = new Statistics();           // collect statistics
    interrupt = new Interrupt;          // start up interrupt
handling
    scheduler = new Scheduler(type);    // initialize the
ready queue
    alarm = new Alarm(randomSlice);     // start up time
slicing

    // We didn't explicitly allocate the current thread we are
running in.
    // But if it ever tries to give up the CPU, we better have
a Thread
    // object to save its state.
    currentThread = new Thread("main");
    currentThread->setStatus(RUNNING);

    interrupt->Enable();
}
```

在`thread.h`中加入需要的method和變數，並於`thread.cc`實作，再去call 測試程式

```
// code/threads/thread.h
class Thread {
  ...
  public:
   ...
    void setBurstTime(int t)    {burstTime = t;}
    int getBurstTime()      {return burstTime;}
    void setStartTime(int t)    {startTime = t;}
    int getStartTime()      {return startTime;}
    void setPriority(int t) {execPriority = t;}
    int getPriority()       {return execPriority;}
    static void SchedulingTest();
  private:
    // some of the private data for this class is listed above
    int burstTime;  // predicted burst time
    int startTime;  // the start time of the thread
    int execPriority;   // the execute priority of the thread
    ...
};
```

```
// code/threads/thread.cc
void
threadBody() {
    Thread *thread = kernel->currentThread;
    while (thread->getBurstTime() > 0) {
        thread->setBurstTime(thread->getBurstTime() - 1);
        kernel->interrupt->OneTick();
        printf("%s: remaining %d\n", kernel->currentThread-
>getName(), kernel->currentThread->getBurstTime());
    }
}
void
Thread::SchedulingTest()
{
    const int thread_num = 4;
    char *name[thread_num] = {"A", "B", "C", "D"};
    int thread_priority[thread_num] = {9, 3, 4, 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();
}
```

```
// code/threads/kernel.cc
void
ThreadedKernel::SelfTest() {
   ...
   currentThread->SelfTest();   // test thread switching
   Thread::SchedulingTest();
   ...
}
```

再來是弄排程的部分，發現`Scheduler`決定thread執行的先後順序的方法，就是根據`Scheduler`的`readyList`中的thread順序，而Priority, SJF, FCFS分別是根據priority, CPU burst time, start time的大小決定先後順序，那就需要`Sortlisted`和compared function了

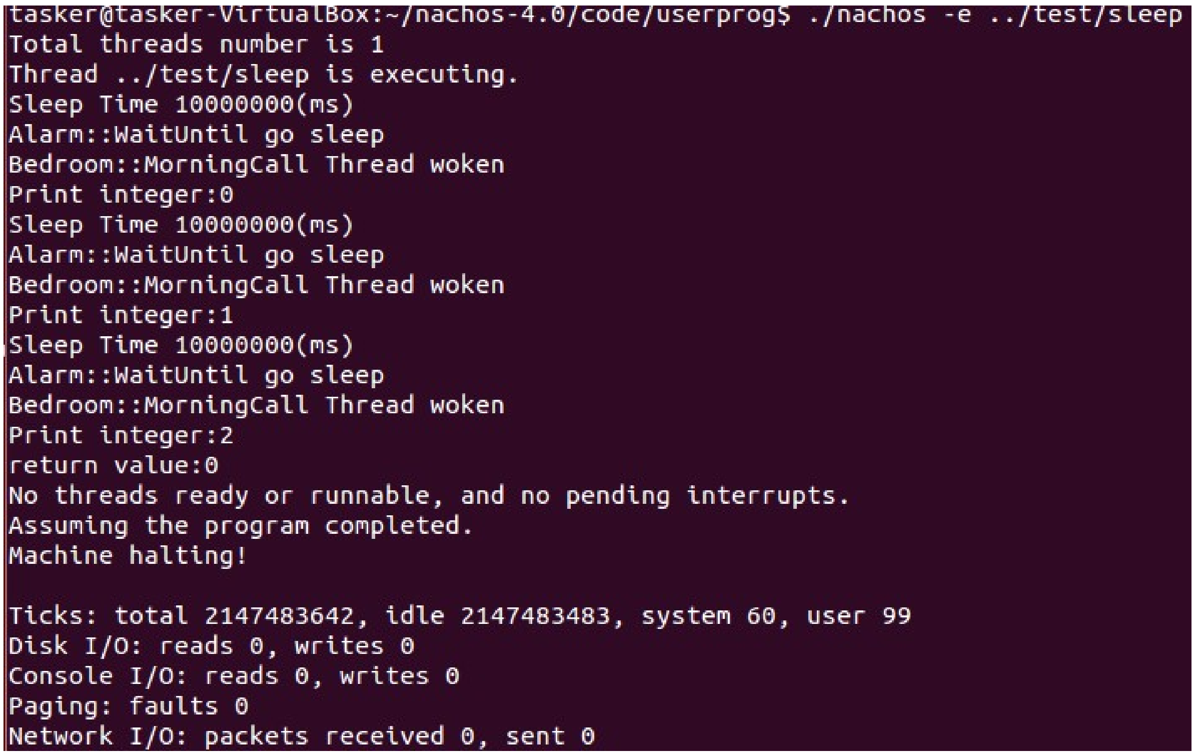
```
// code/threads/scheduler.cc
int SJFCompare(Thread *a, Thread *b) {
    if(a->getBurstTime() == b->getBurstTime())
        return 0;
    return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
}
int PriorityCompare(Thread *a, Thread *b) {
    if(a->getPriority() == b->getPriority())
        return 0;
    return a->getPriority() > b->getPriority() ? 1 : -1;
}
int FIFOCompare(Thread *a, Thread *b) {
    return 1;
}
//--------------------------------------------------------------
----------
// Scheduler::Scheduler
//  Initialize the list of ready but not running threads.
//  Initially, no ready threads.
//--------------------------------------------------------------
----------
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;
}
```

如果需要(Preemptive)，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

```
// code/threads/alarm.cc
void
Alarm::CallBack()
{
    ...
    bool woken = _bedroom.MorningCall();
    ...
    else {          //preempt
    if(kernel->scheduler->getSchedulerType() == RR ||
        kernel->scheduler->getSchedulerType() == Priority ) {
    cout << "=== interrupt->YieldOnReturn ===" << endl;
    }
}
```

## Result

### Experiment result - Sleep

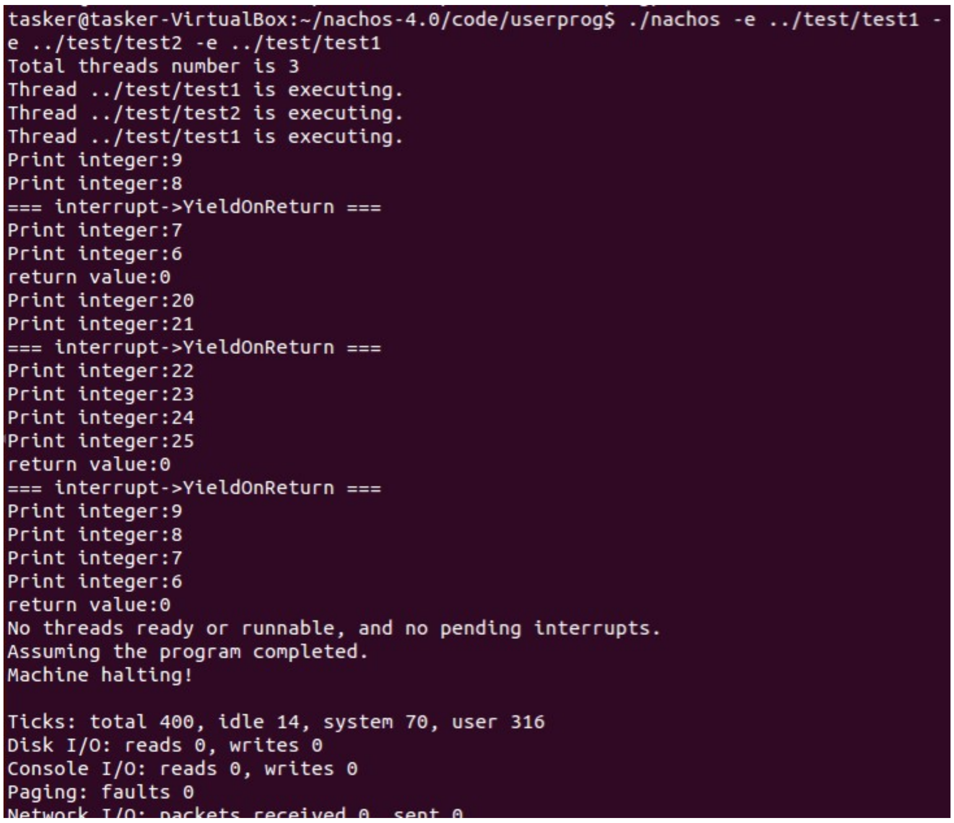


```
tasker@tasker-VirtualBox:~/nachos-4.0/code/userprog$ ./nachos -e ../test/sleep
Total threads number is 1
Thread ../test/sleep is executing.
Sleep Time 10000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:0
Sleep Time 10000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:1
Sleep Time 10000000(ms)
Alarm::WaitUntil go sleep
Bedroom::MorningCall Thread woken
Print integer:2
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2147483642, idle 2147483483, system 60, user 99
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
```

## Experiment result - Scheduling

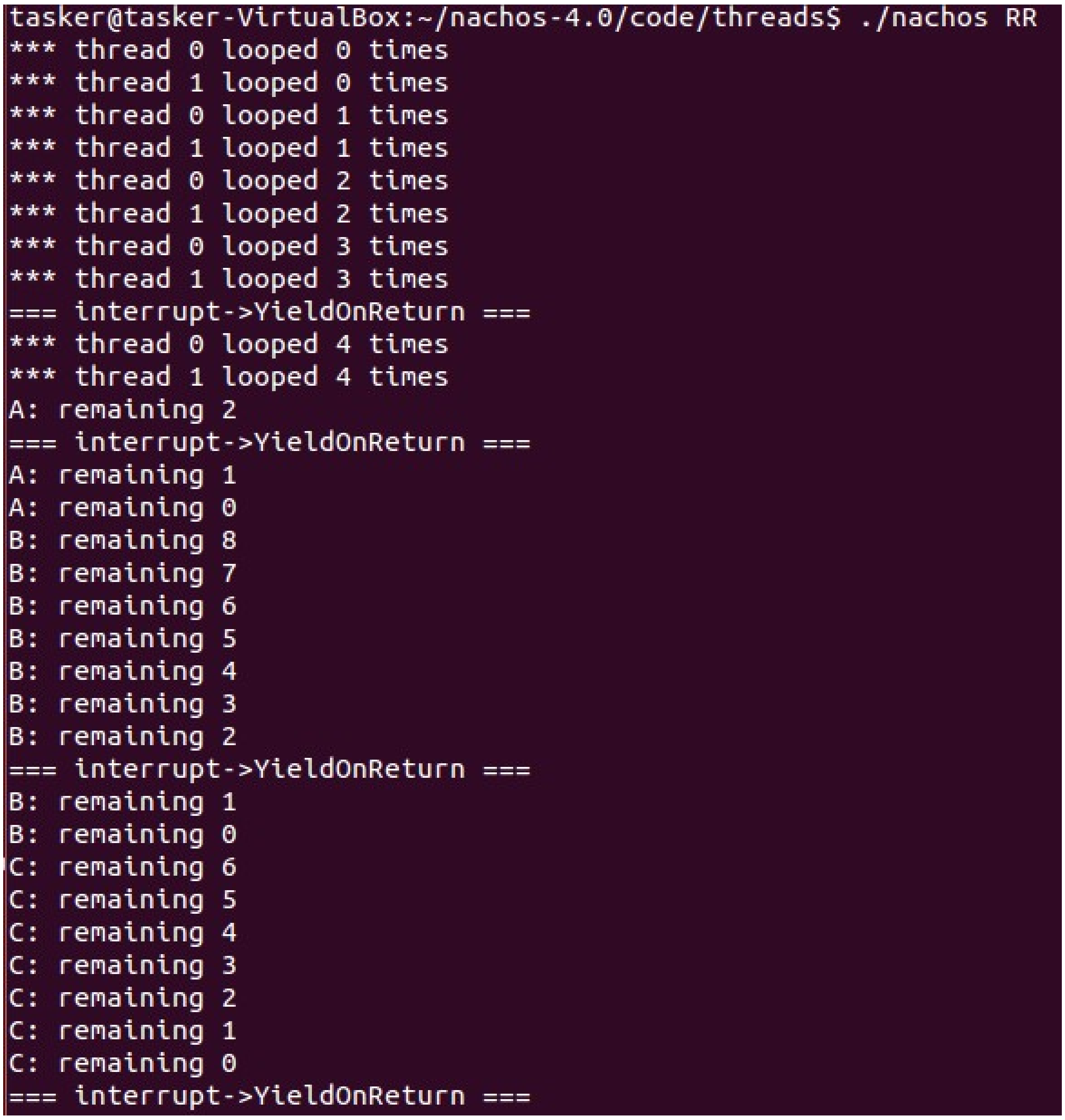
`$./nachos -e ../test/test1 -e ../test/test2 -e ../test/test1`

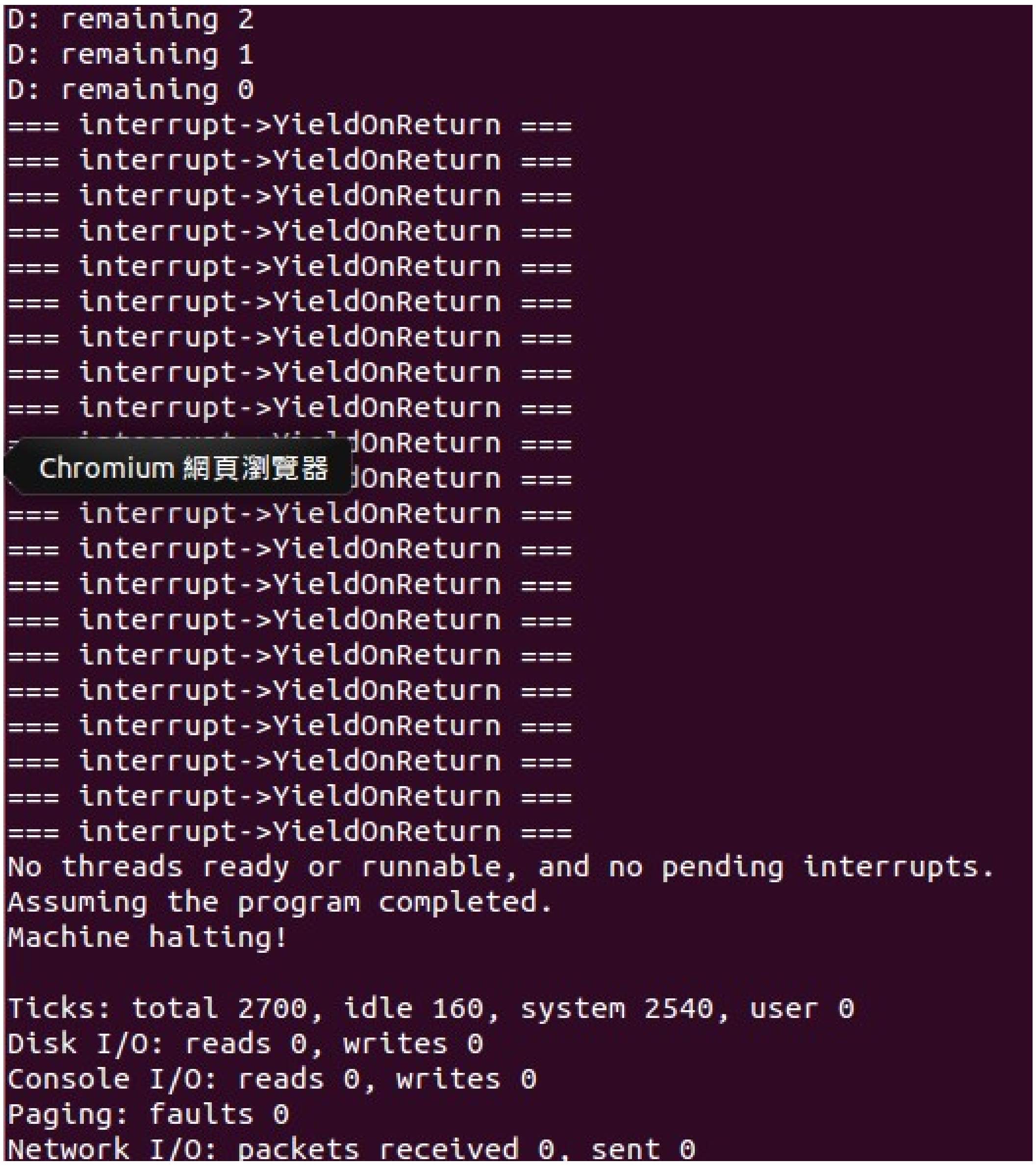


```
tasker@tasker-VirtualBox:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -
e ../test/test2 -e ../test/test1
Total threads number is 3
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Thread ../test/test1 is executing.
Print integer:9
Print integer:8
=== interrupt->YieldOnReturn ===
Print integer:7
Print integer:6
return value:0
Print integer:20
Print integer:21
=== interrupt->YieldOnReturn ===
Print integer:22
Print integer:23
Print integer:24
Print integer:25
return value:0
=== interrupt->YieldOnReturn ===
Print integer:9
Print integer:8
Print integer:7
Print integer:6
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 400, idle 14, system 70, user 316
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
```

- RR

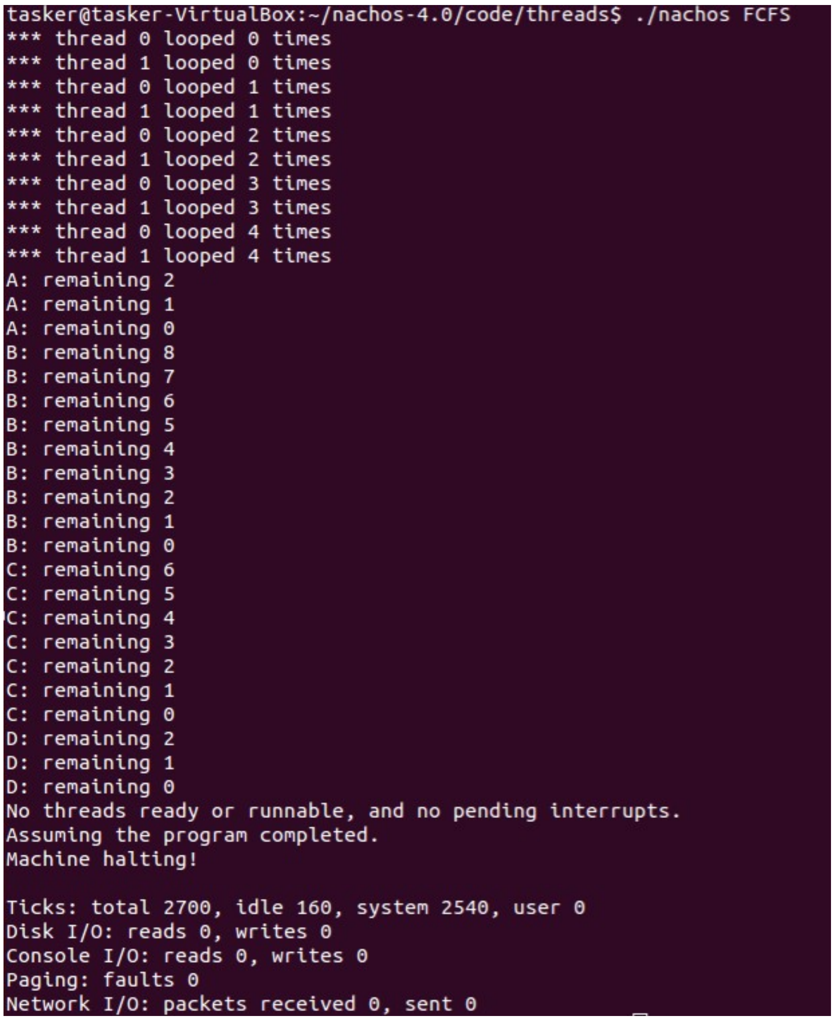

```
tasker@tasker-VirtualBox:~/nachos-4.0/code/threads$ ./nachos RR
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
=== interrupt->YieldOnReturn ===
*** thread 0 looped 4 times
*** thread 1 looped 4 times
A: remaining 2
=== interrupt->YieldOnReturn ===
A: 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
=== interrupt->YieldOnReturn ===
B: 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
=== interrupt->YieldOnReturn ===
```



```
D: remaining 2
D: remaining 1
D: remaining 0
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=                      dOnReturn ===
                       dOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
No 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
Paging: faults 0
Network I/O: packets received 0, sent 0
```

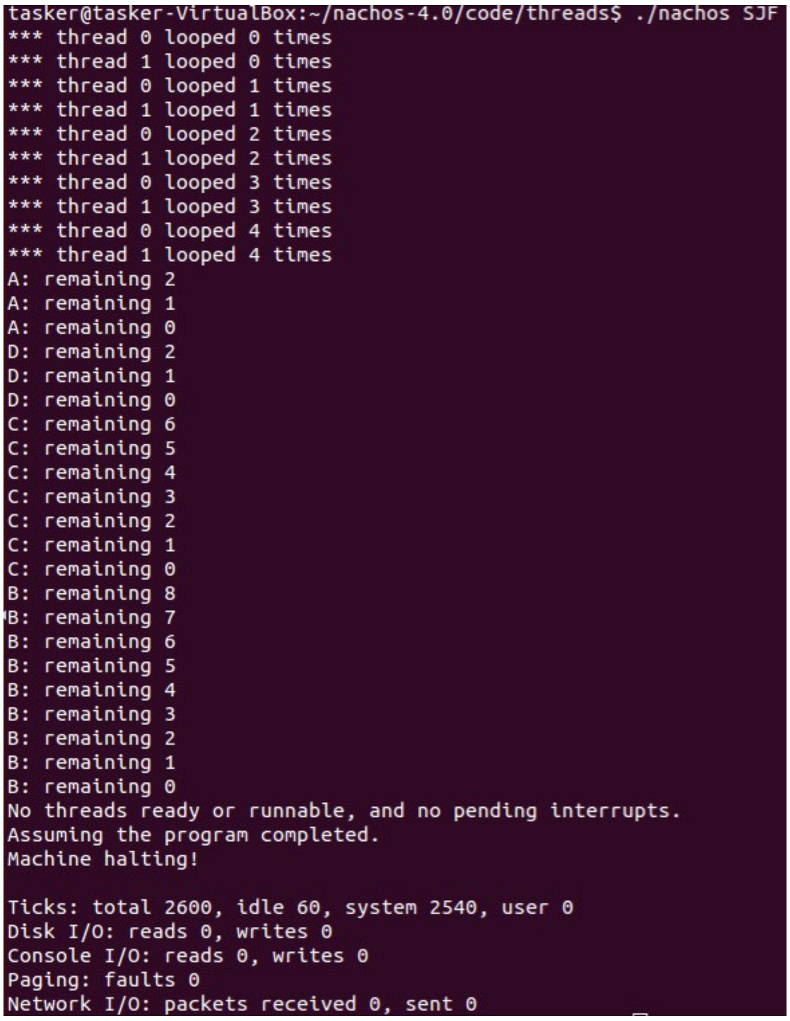
- FCFS



```
tasker@tasker-VirtualBox:~/nachos-4.0/code/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
*** thread 0 looped 4 times
*** thread 1 looped 4 times
A: remaining 2
A: 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
B: 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
D: remaining 2
D: remaining 1
D: remaining 0
No 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
Paging: faults 0
Network I/O: packets received 0, sent 0
```

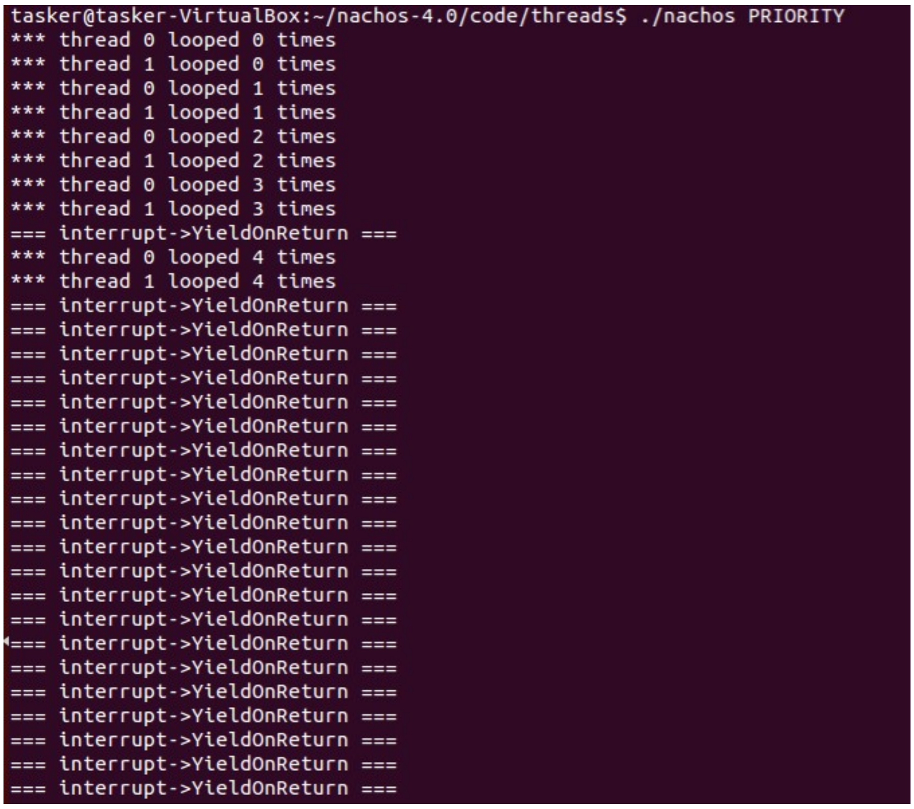
- SJF



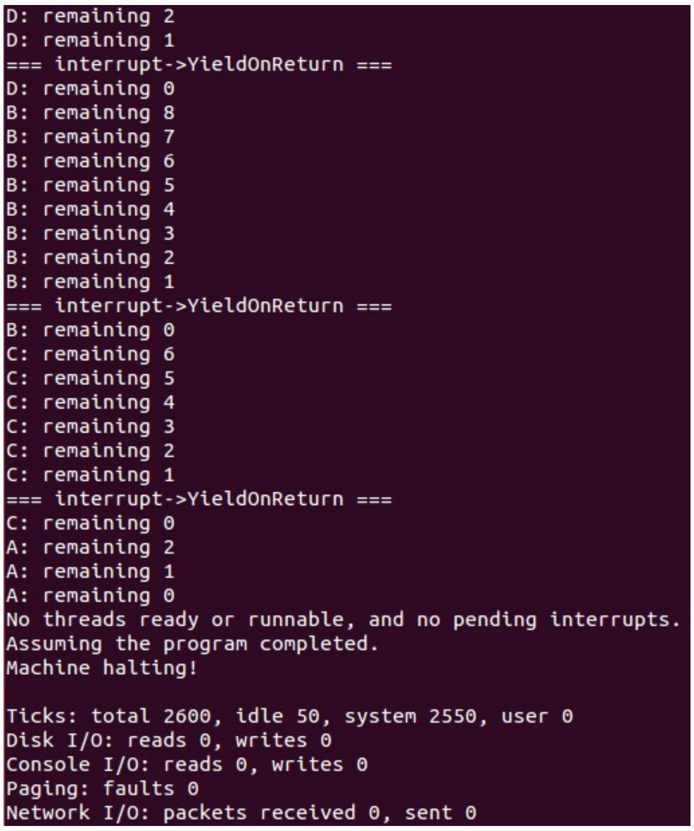
```
tasker@tasker-VirtualBox:~/nachos-4.0/code/threads$ ./nachos SJF
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
A: remaining 2
A: remaining 1
A: remaining 0
D: remaining 2
D: remaining 1
D: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
B: remaining 0
No 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
```

- Priority



```
tasker@tasker-VirtualBox:~/nachos-4.0/code/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
=== interrupt->YieldOnReturn ===
*** thread 0 looped 4 times
*** thread 1 looped 4 times
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
```



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

Ticks: total 2600, idle 50, system 2550, 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
```

可以看出這幾個排程都有符合`SchedulingTest()`的參數做執行，`FIFO`是依序執行，順序是ABCD，`SJF`是依照`thread_burst`由小到大執行，順序是ADCB，`PRIORITY`是依照`thread_priority`由小到大執行，順序是DBCA。

## Extra effort or observation

- 在做完第二題時因為要看是否有interupt，所以會輸出`interrupt->YieldOnReturn`，但是這會造成終端機執行第一題時做無窮迴圈，因此要註解掉。

```
void 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()) {// is it time to quit?
        if (!interrupt->AnyFutureInterrupts()) {
        timer->Disable();    // turn off the timer
    }
    } else {              //preempt
    if(kernel->scheduler->getSchedulerType() == RR ||
        kernel->scheduler->getSchedulerType() == Priority ) {
```

```
     //cout << "=== interrupt->YieldOnReturn ===" << endl;
    //need comment when execute sleep.c
     }
 }
 }
```

- 在修改Makefile時，縮排不能用空白鍵，而是要用tab，不然會報錯。