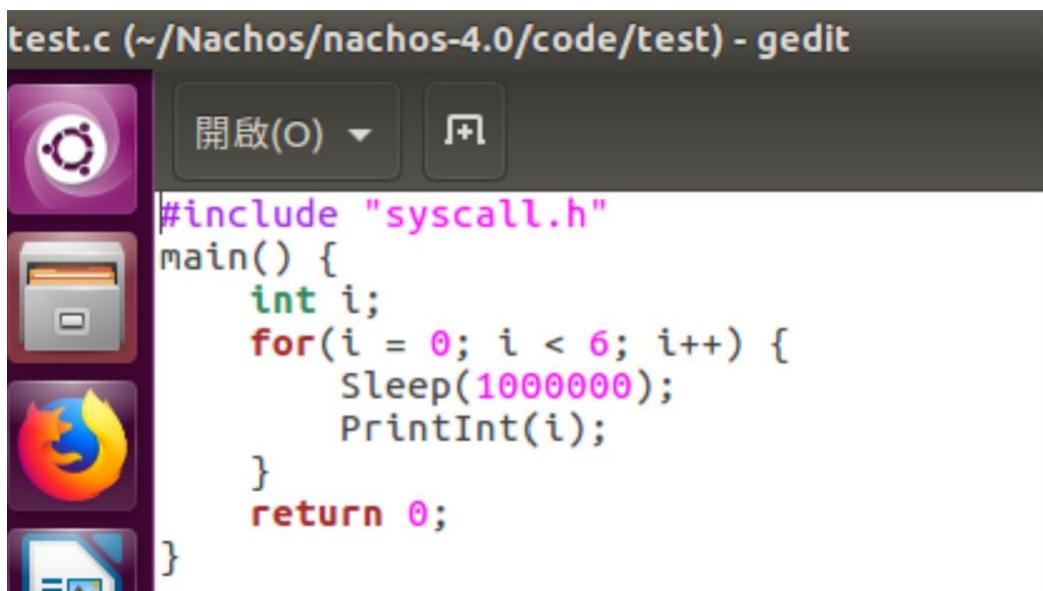


2023 FALL OS Project 2 System Call & CPU Scheduling

R12631055 林東甫

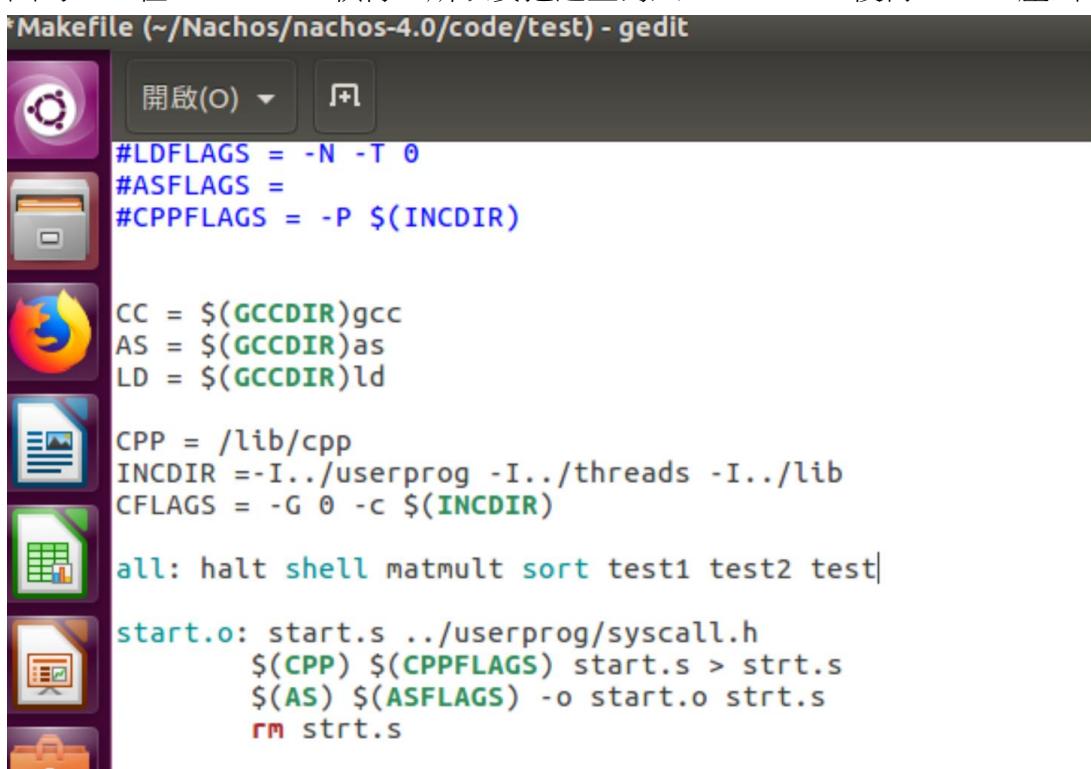
Part1:

System call 是程式跟 OS 操作互動的一種方法，我們經常透過程式來發出 system call 好對 OS kernel 做一些 request，而這次實作的目標就是做出一個 Sleep function 讓 Thread 進入休眠。首先，先寫出這次的 test program : test.c



```
#include "syscall.h"
main() {
    int i;
    for(i = 0; i < 6; i++) {
        Sleep(10000000);
        PrintInt(i);
    }
    return 0;
}
```

因為 test 在 nachOS 上執行，所以要把這些寫入 Makefile ，後再 make 產出。



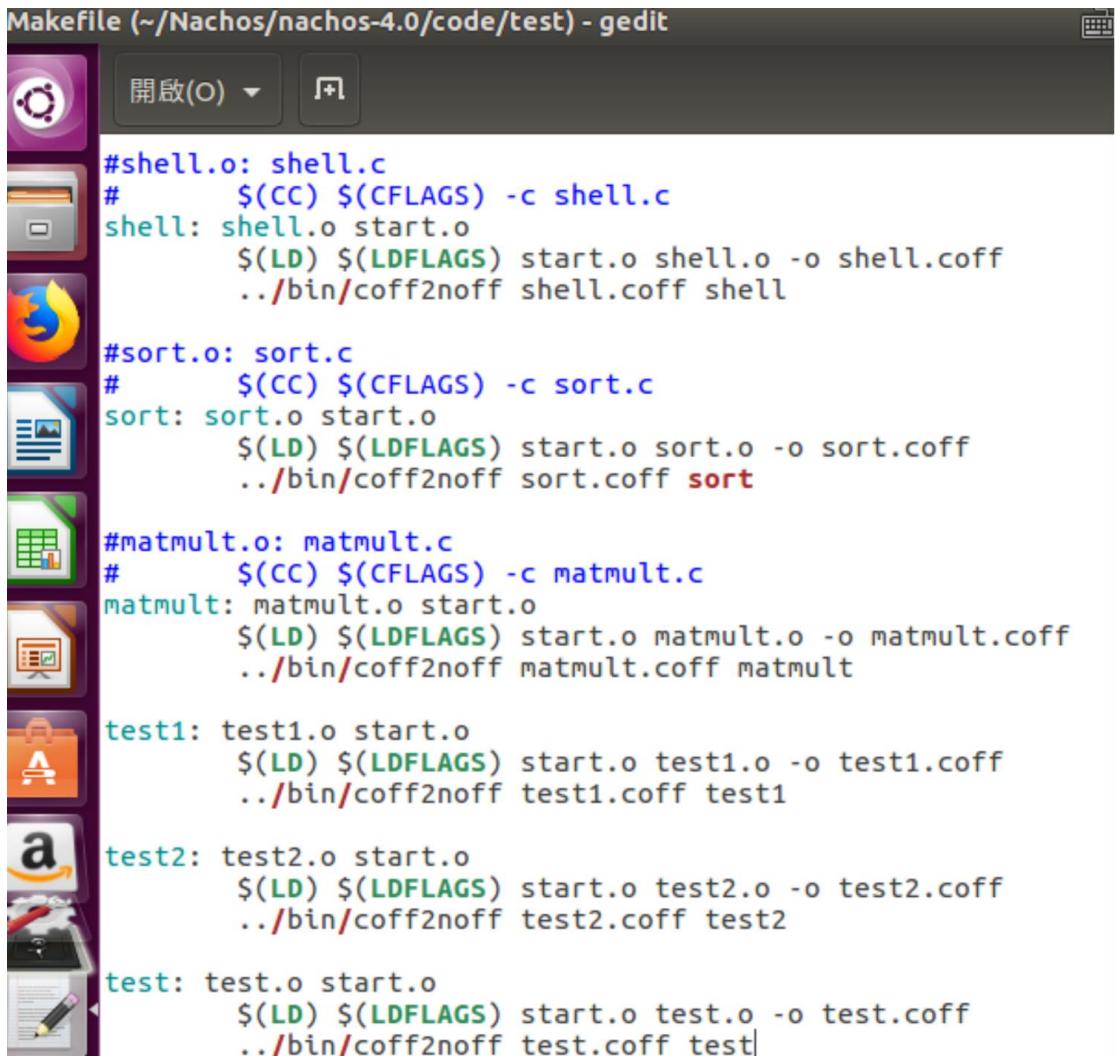
```
#LDFLAGS = -N -T 0
#ASFLAGS =
#CPPFLAGS = -P $(INCDIR)

CC = $(GCCDIR)gcc
AS = $(GCCDIR)as
LD = $(GCCDIR)ld

CPP = /lib/cpp
INCDIR = -I../userprog -I../threads -I../lib
CFLAGS = -G 0 -c $(INCDIR)

all: halt shell matmult sort test1 test2 test

start.o: start.s ../userprog/syscall.h
$(CPP) $(CPPFLAGS) start.s > strt.s
$(AS) $(ASFLAGS) -o start.o strt.s
rm strt.s
```



```
Makefile (~/Nachos/nachos-4.0/code/test) - gedit
開啟(O) ▾
```

```
#shell.o: shell.c
#      $(CC) $(CFLAGS) -c shell.c
shell: shell.o start.o
        $(LD) $(LDFLAGS) start.o shell.o -o shell.coff
        ..../bin/coff2noff shell.coff shell

#sort.o: sort.c
#      $(CC) $(CFLAGS) -c sort.c
sort: sort.o start.o
        $(LD) $(LDFLAGS) start.o sort.o -o sort.coff
        ..../bin/coff2noff sort.coff sort

#matmult.o: matmult.c
#      $(CC) $(CFLAGS) -c matmult.c
matmult: matmult.o start.o
        $(LD) $(LDFLAGS) start.o matmult.o -o matmult.coff
        ..../bin/coff2noff matmult.coff matmult

test1: test1.o start.o
        $(LD) $(LDFLAGS) start.o test1.o -o test1.coff
        ..../bin/coff2noff test1.coff test1

test2: test2.o start.o
        $(LD) $(LDFLAGS) start.o test2.o -o test2.coff
        ..../bin/coff2noff test2.coff test2

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

因為 Nachos 是使用 MIPS 的指令集，所以必須用 MIPS assembling language 來實作，整體來上可以直接參考 PrintInt 的結構：

```
PrintInt:
    addiu   $2,$0,SC_PrintInt
    syscall
    j       $31
    .end    PrintInt

    .globl1 Sleep
    .ent    Sleep

Sleep:
    addiu   $2,$0,SC_Sleep
    syscall
    j       $31
    .end    Sleep|
```

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

__main:
    j       $31
```

宣告也是模仿 PrintInt

```
#include "copyright.h"

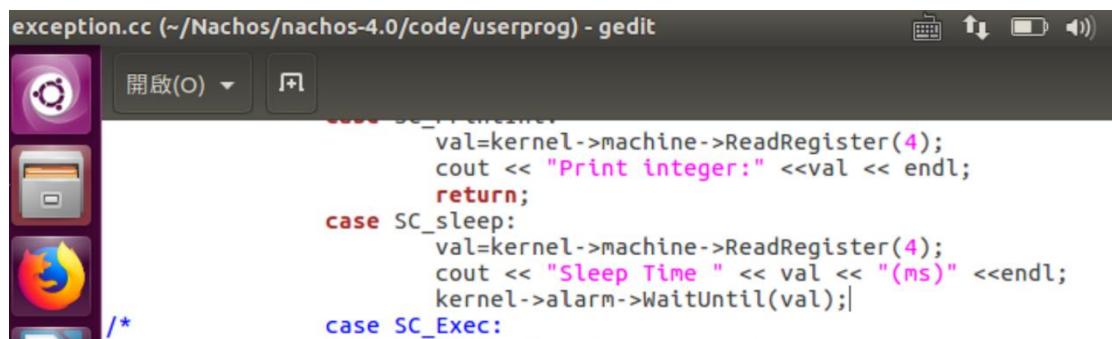
/* system call codes -- used by t
 * is being asked for
 */
#define SC_Halt      0
#define SC_Exit      1
#define SC_Exec      2
#define SC_Join      3
#define SC_Create    4
#define SC_Open       5
#define SC_Read       6
#define SC_Write      7
#define SC_Close      8
#define SC_ThreadFork 9
#define SC_ThreadYield 10
#define SC_PrintInt   11
#define SC_Sleep      12

#ifndef IN_ASM

/* Yield the CPU to another runnable thread, whether
 * or not.
 */
void ThreadYield();

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

以及例外操作：



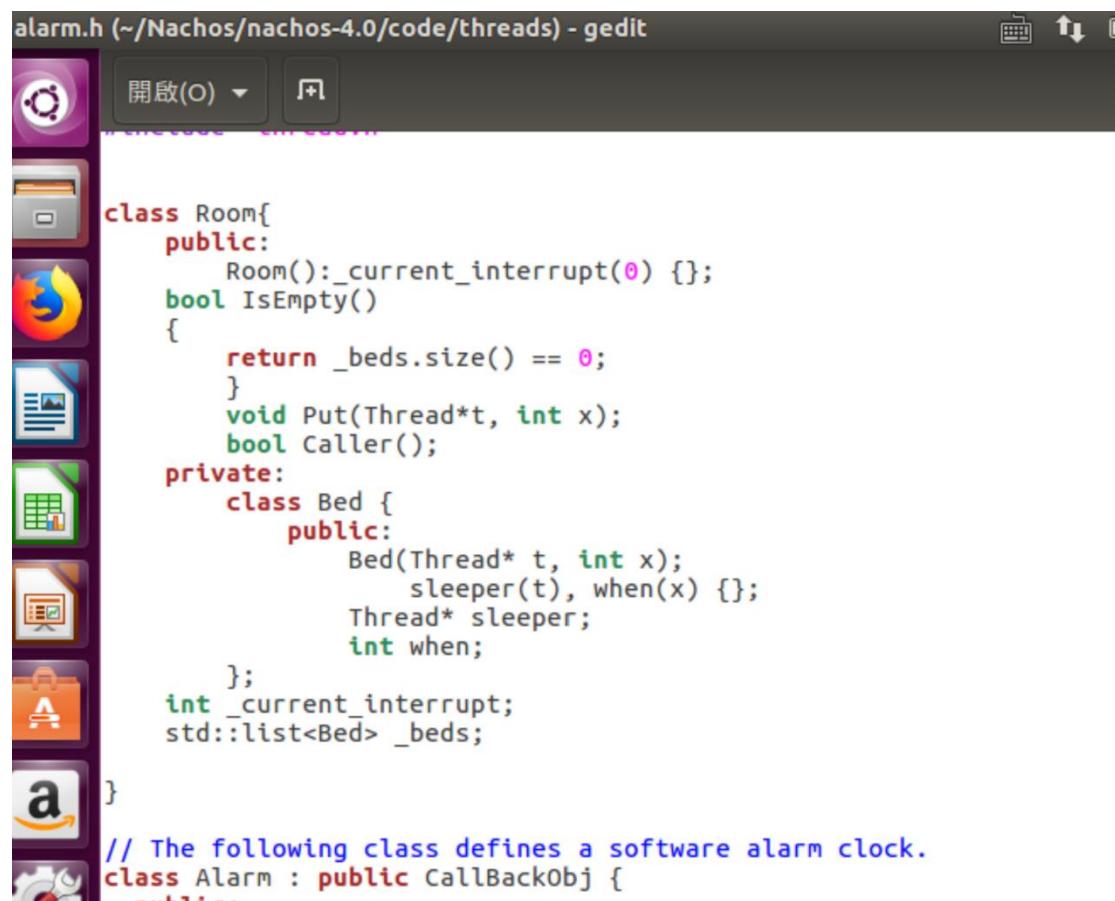
The screenshot shows a terminal window titled "exception.cc (~/Nachos/nachos-4.0/code/userprog) - gedit". The code inside the terminal is as follows:

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

剛剛例外操作中看到的 kernel 中有 alarm，也就是每隔一段時間，就會呼叫 Alarm::CallBack()，因此，對這個鬧鐘設置個累加器 room::_current_interrupt，全局去記數，作為毫秒來看待，每加一次就相當於過了 1 ms。

假設現在某一個 time = _current_interrupt 呼叫 Sleep(x) 後，將 (Thread address, _current_interrupt + x) 丟入 List，則期望在 _current_interrupt + x 的時候甦醒。因此，每當 _current_interrupt++ 就去檢查 List 中，誰的預期時間小於 _current_interrupt，就將其從 List 中清除。

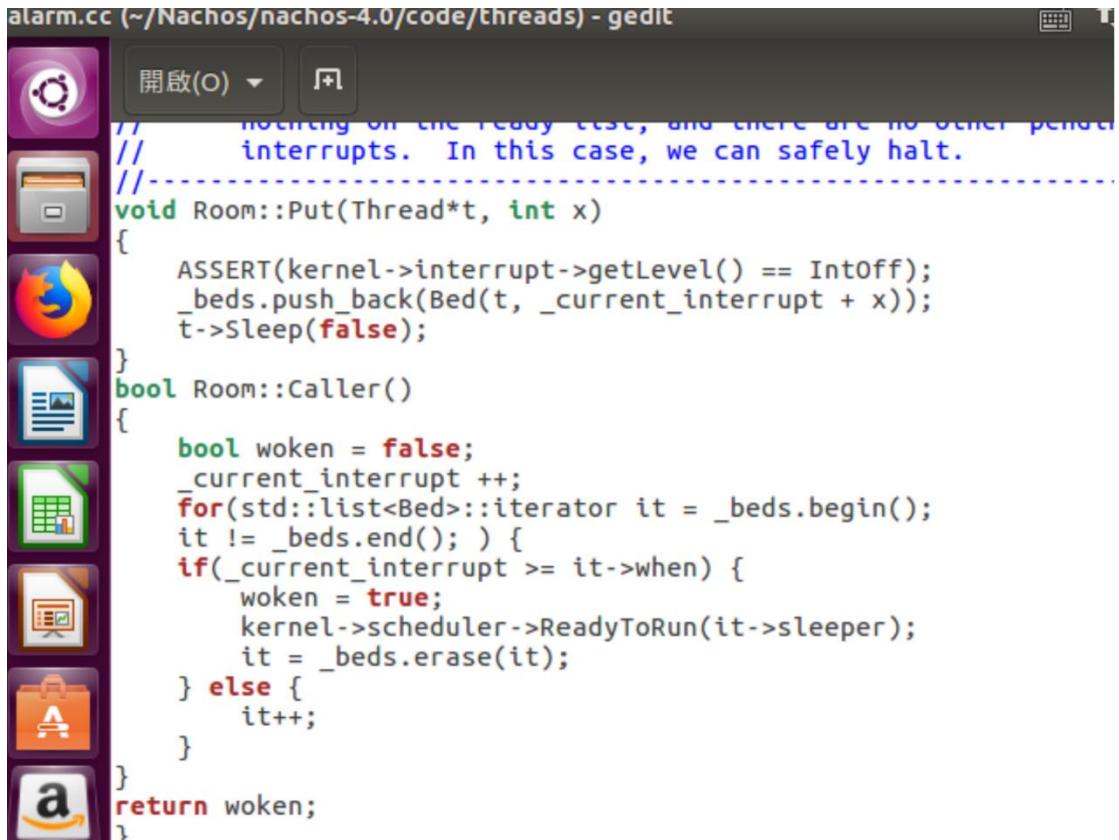
當有程序呼叫 Sleep() 時，會呼叫 WaitUntil()，然後將其丟入 room 安眠。然後在 CallBlack() 被呼叫時，來去 room 檢查誰該醒來了。



The screenshot shows a Gedit text editor window with the file 'alarm.h' open. The code defines a Room class with a list of beds and an alarm clock.

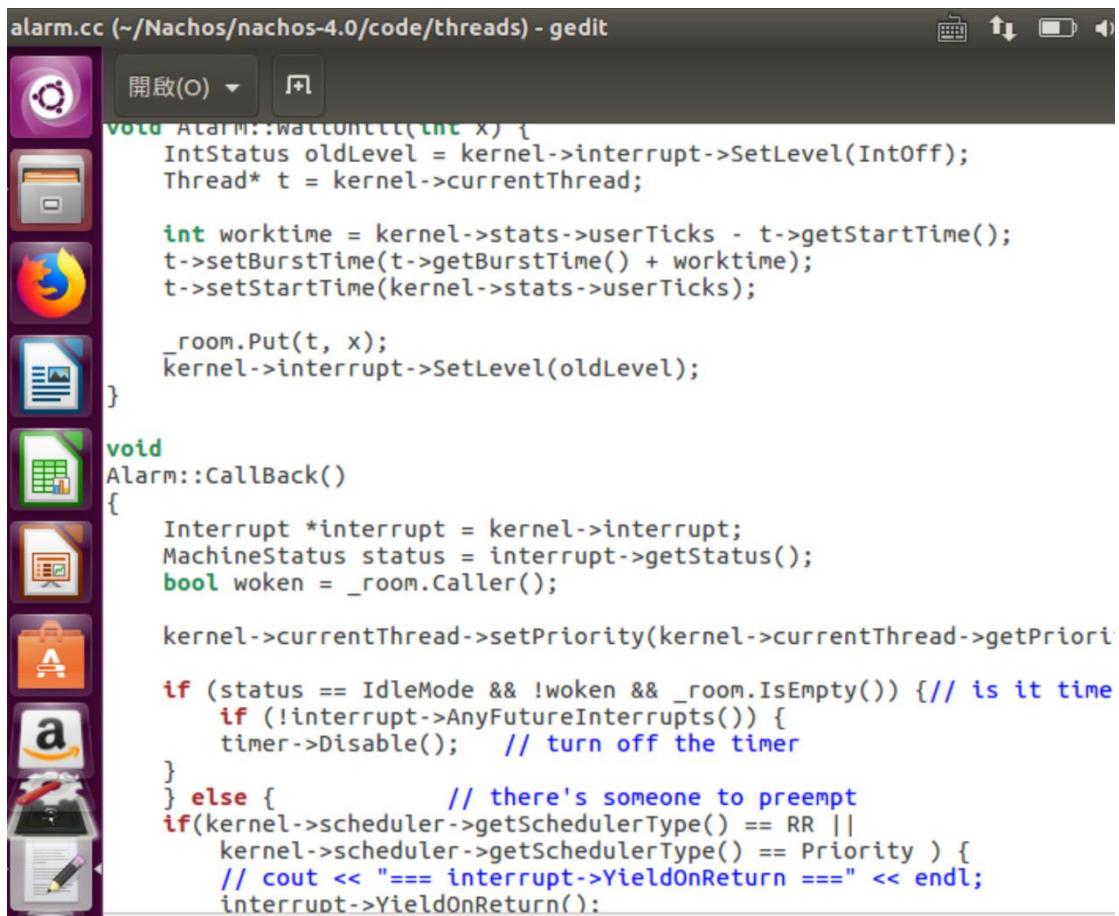
```
alarm.h (~/Nachos/nachos-4.0/code/threads) - gedit
開啟(O) +
```

```
class Room{
public:
    Room():_current_interrupt(0) {};
    bool IsEmpty()
    {
        return _beds.size() == 0;
    }
    void Put(Thread*t, int x);
    bool Caller();
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 {
...L1...
```



```
//      nothing on the ready list, and there are no other pending
//      interrupts. In this case, we can safely halt.
//-----
void Room::Put(Thread*t, int x)
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    _beds.push_back(Bed(t, _current_interrupt + x));
    t->Sleep(false);
}
bool Room::Caller()
{
    bool woken = false;
    _current_interrupt++;
    for(std::list<Bed>::iterator it = _beds.begin();
        it != _beds.end(); ) {
        if(_current_interrupt >= it->when) {
            woken = true;
            kernel->scheduler->ReadyToRun(it->sleeper);
            it = _beds.erase(it);
        } else {
            it++;
        }
    }
    return woken;
}
```

接著在 alarm 實作 waituntil 和 callback :



```
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);

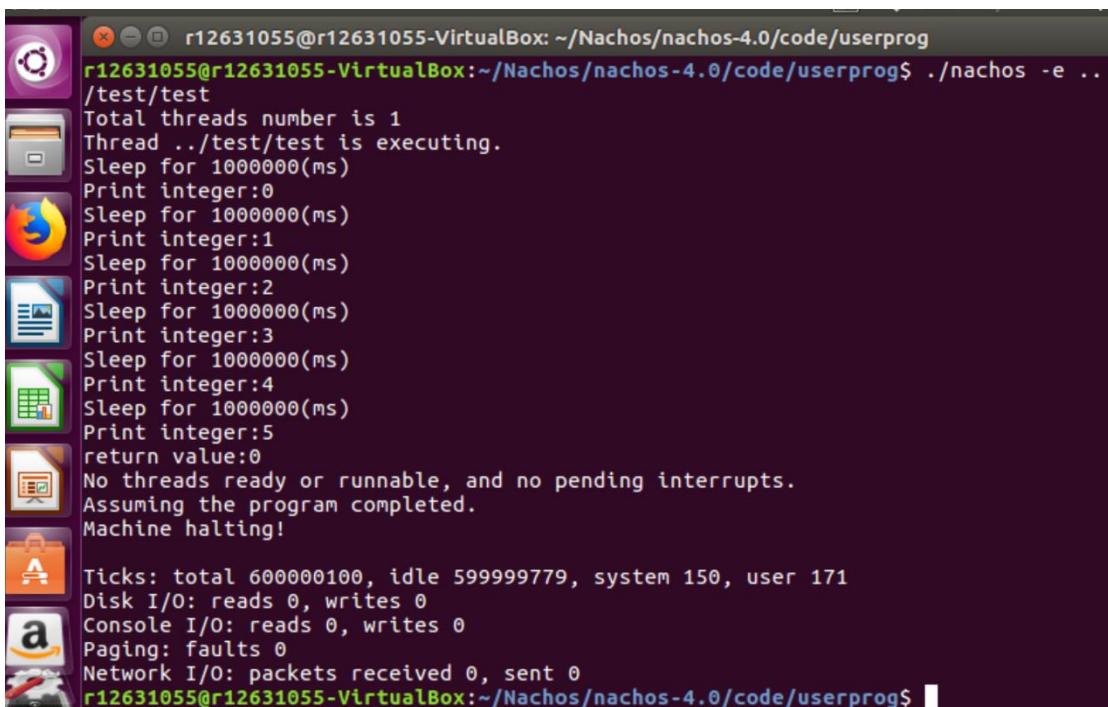
    room.Put(t, x);
    kernel->interrupt->SetLevel(oldLevel);
}

void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool woken = _room.Caller();

    kernel->currentThread->setPriority(kernel->currentThread->getPriori

    if (status == IdleMode && !woken && _room.IsEmpty()) {// is it time
        if (!interrupt->AnyFutureInterrupts()) {
            timer->Disable(); // turn off the timer
        }
    } else { // there's someone to preempt
        if(kernel->scheduler->getSchedulerType() == RR ||
            kernel->scheduler->getSchedulerType() == Priority ) {
            // cout << "== interrupt->YieldOnReturn ==" << endl;
            interrupt->YieldOnReturn();
        }
    }
}
```

最後實際測試我們寫的 test program。



The screenshot shows a terminal window on a Linux desktop. The title bar indicates the session is on a VirtualBox machine with the identifier r12631055. The command being run is ./nachos -e .. /test/test. The terminal output shows a sequence of threads being created and executed, each printing an integer value from 0 to 5 and sleeping for 1000000ms. After all threads have completed, it prints a message indicating no threads are ready or runnable, the program is completed, and the machine is halting. At the bottom, system statistics are displayed, including ticks, disk I/O, console I/O, paging, and network I/O.

```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$ ./nachos -e .. /test/test
Total threads number is 1
Thread .. /test/test is executing.
Sleep for 1000000(ms)
Print integer:0
Sleep for 1000000(ms)
Print integer:1
Sleep for 1000000(ms)
Print integer:2
Sleep for 1000000(ms)
Print integer:3
Sleep for 1000000(ms)
Print integer:4
Sleep for 1000000(ms)
Print integer:5
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
Ticks: total 600000100, idle 599999779, system 150, user 171
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
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$
```

其實在 makefile 的時候，還對 make 的掌握不太嫻熟，花了很多時間上網研究資料以及實作最後才得以成功。

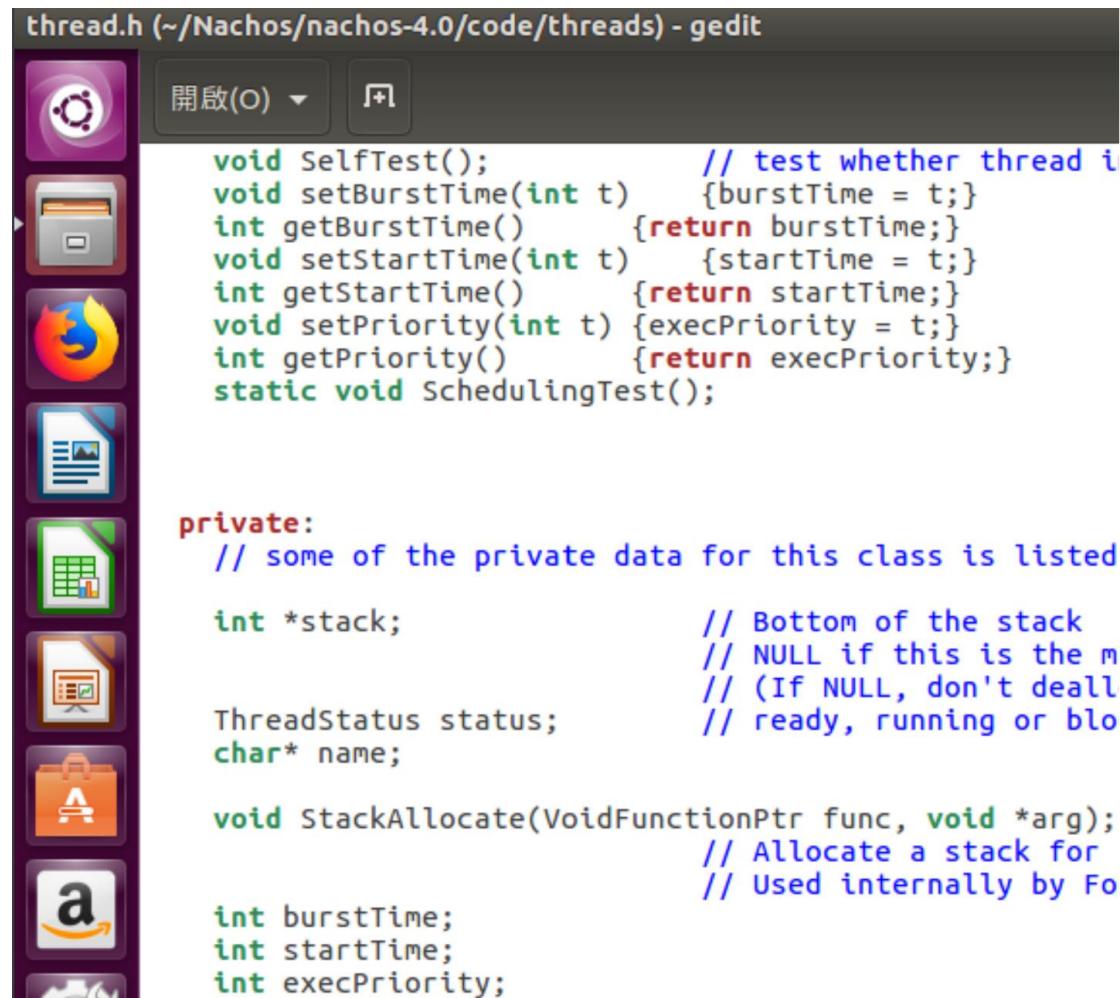
當我們在開發中涉及眾多檔案時，常常會引起一些問題。如果影響只有兩三個文件，那麼修改程式後直接重新編譯全部文件就行，但如果影響的文件較多，這種簡單的處理方式就有問題了。在開發新的程式也是如此，如果我們只建立或修改一個文件，卻要重新建立或編譯所有文件，那麼顯然會浪費很多時間。因此要利用 make 來幫我們解決這些問題。

Part2

主要目標是透過實作各種 CPU 排程算法來了解 CPU 排程，基本上我們會實做：

1. First-Come-First-Service (FCFS)
2. Shortest-Job-First (SJF)
3. Priority
4. Round-Robin

首先先增加 setPriority, setBurstTime, SchedulingTest 等 method header and body。



```
thread.h (~/Nachos/nachos-4.0/code/threads) - gedit

void SelfTest();           // test whether thread is
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
int *stack;                // Bottom of the stack
                           // NULL if this is the main
                           // (If NULL, don't deallocate)
ThreadStatus status;        // ready, running or blocked
char* name;

void StackAllocate(VoidFunctionPtr func, void *arg);
                           // Allocate a stack for
                           // Used internally by
                           // CreateThread
int burstTime;
int startTime;
int execPriority;
```

接著在 main 裡面做個可選擇任一算法的判斷式：

```
DEBUG(dbgThread, "Entering main");

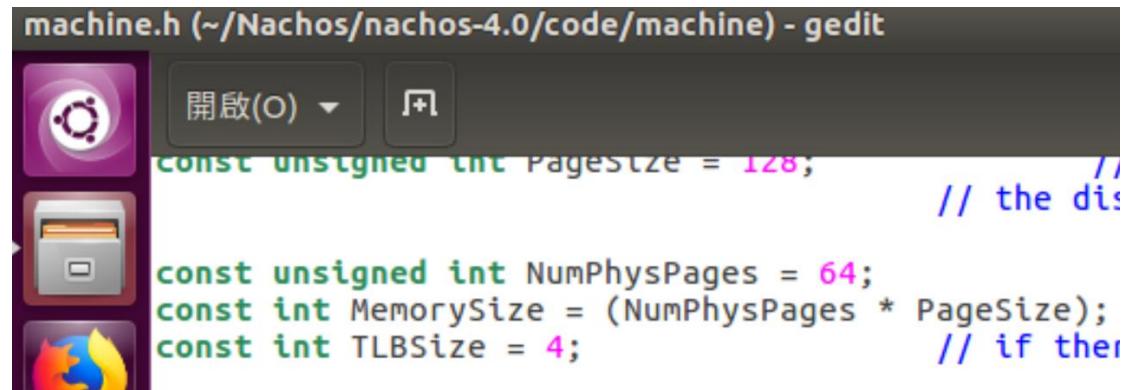
SchedulerType type = RR;
if(strcmp(argv[1], "FCFS") == 0) {
    type = FIFO;
} else if (strcmp(argv[1], "SJF") == 0) {
    type = SJF;
} else if (strcmp(argv[1], "PRIORITY") == 0) {
    type = Priority;
} else if (strcmp(argv[1], "RR") == 0) {
    type = RR;
}

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

CallOnUserAbort(Cleanup); // if user hits ctl-C

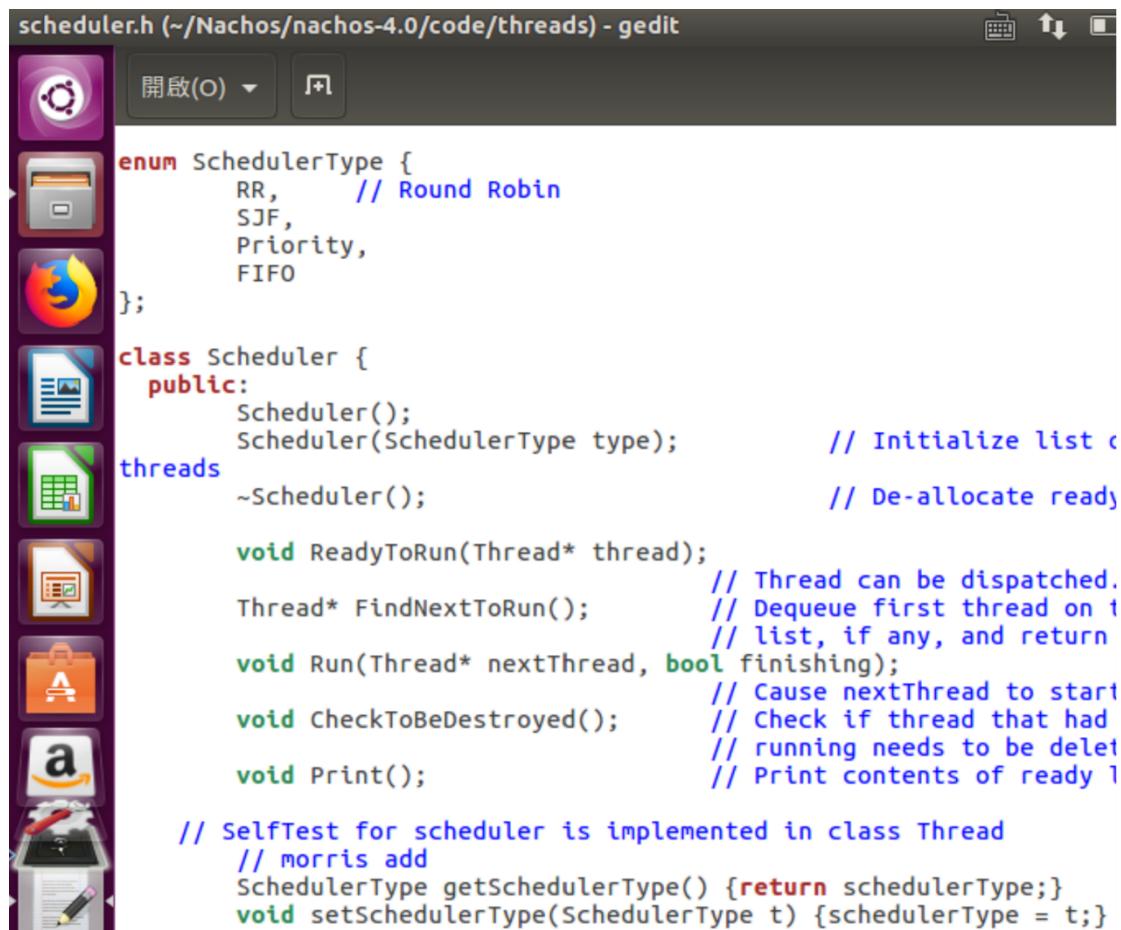
kernel->SelfTest();
kernel->Run();
```

視情況有可能會需要因此把記憶體調高一些：



```
machine.h (~/Nachos/nachos-4.0/code/machine) - gedit
const unsigned int PageSize = 128; // the size of each page
const unsigned int NumPhysPages = 64;
const int MemorySize = (NumPhysPages * PageSize);
const int TLBSize = 4; // if there are 4 entries in the TLB
```

使用多型然後決定要用哪一種類型排程，並且宣告相對應的 compare function：



```
scheduler.h (~/Nachos/nachos-4.0/code/threads) - gedit
调度器类型枚举和调度器类的头文件。调度器类包含调度方法（ReadyToRun, FindNextToRun, Run, CheckToBeDestroyed, Print）和调度类型（RR, SJF, Priority, FIFO）。
```

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

class Scheduler {
public:
    Scheduler();
    Scheduler(SchedulerType type);           // Initialize list of threads
    ~Scheduler();                           // De-allocate ready queue
    void ReadyToRun(Thread* thread);        // Thread can be dispatched.
    Thread* FindNextToRun();               // Dequeue first thread on ready
                                         // list, if any, and return
    void Run(Thread* nextThread, bool finishing); // Cause nextThread to start
    void CheckToBeDestroyed();             // Check if thread that had
                                         // been running needs to be deleted
    void Print();                         // Print contents of ready queue

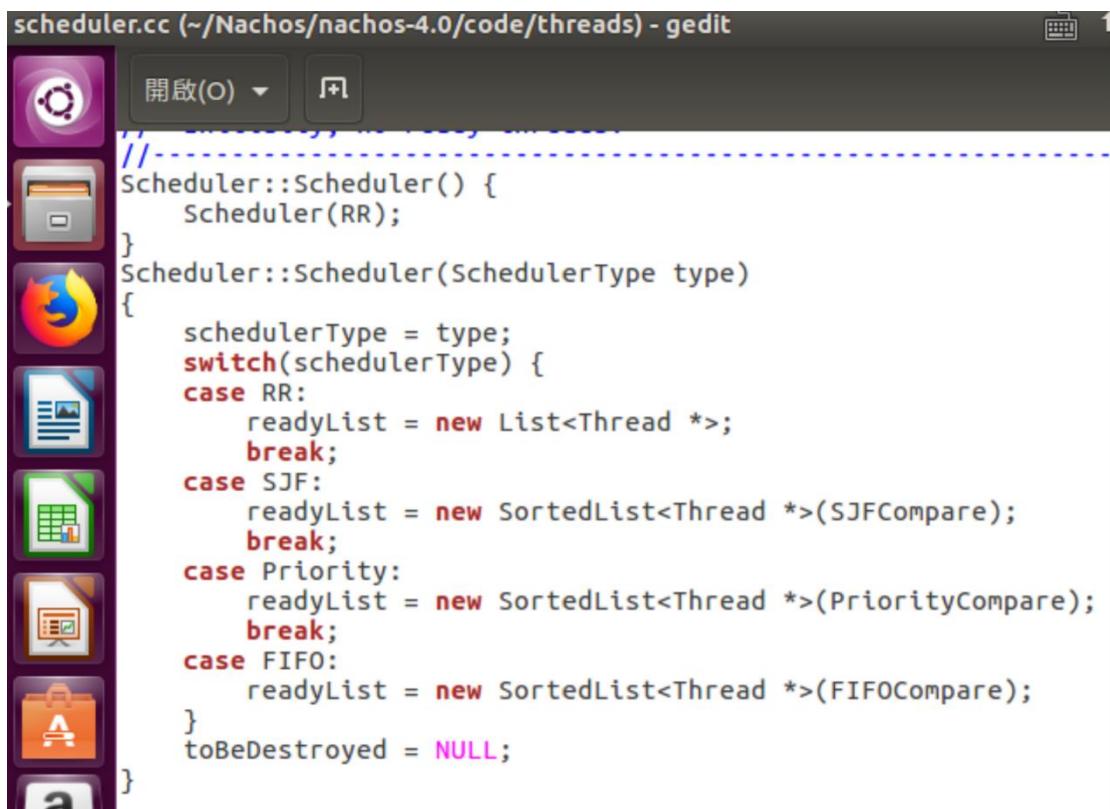
    // SelfTest for scheduler is implemented in class Thread
    // morris add
    SchedulerType getSchedulerType() {return schedulerType;}
    void setSchedulerType(SchedulerType t) {schedulerType = t;}
}
```

設計可以分別比較執行時間長短、Priority 和 burst time 次序：



```
scheduler.cc (~/Nachos/nachos-4.0/code/threads) - gedit
实现三个比较函数：SJFCompare、PriorityCompare 和 FIFOCompare，分别用于比较两个线程的执行时间、优先级和FIFO次序。
```

```
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;
}
```

在 alarm 裡 Callback 和 waituntil 的宣告與實作

```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool woken = _room.Caller();

    kernel->currentThread->setPriority(kernel->currentThread->getPriority() - 1);

    if (status == IdleMode && !woken && _room.IsEmpty()) { // is it time to quit?
        if (!interrupt->AnyFutureInterrupts()) {
            timer->Disable(); // turn off the timer
        }
    } else { // there's someone to preempt
        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;

    int worktime = kernel->stats->userTicks - t->getStartTime();
    t->setBurstTime(t->getBurstTime() + worktime);
    t->setStartTime(kernel->stats->userTicks);

    _room.Put(t, x);
    kernel->interrupt->SetLevel(oldLevel);
}
```

最後在 userkernel 和 netkernel 宣告 initialize function

userkernel.cc (~/Nachos/nachos-4.0/code/userprog) - gedit

```
void UserProgKernel::Initialize()
{
    Initialize(RR);
}
void UserProgKernel::Initialize(SchedulerType type)
{
    ThreadedKernel::Initialize(type); // init multithreading
    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
#ifndef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}
```

netkernel.cc (~/Nachos/nachos-4.0/code/network) - gedit

```
}
*/
void NetKernel::Initialize() {
    Initialize(RR);
}
void NetKernel::Initialize(SchedulerType type)
{
    UserProgKernel::Initialize(type); // init other kernel
    postOfficeIn = new PostOfficeInput(10);
    postOfficeOut = new PostOfficeOutput(reliability, 10);
}
```

我們在之前已經把相對應的格式都設置好了，因此只要輸入想要測試的類型，就能測試我們的 scheduling algorithm。

首先從 FCFS 開始：

```
r12631055@r12631055-VirtualBox:~/Nachos/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
```

```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$ ./nachos FCFS
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
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$
```

能看出遵循的就是簡單的 FIFO 的方式，按照順序的將 ABCD 處理完。

其次是 SJF :

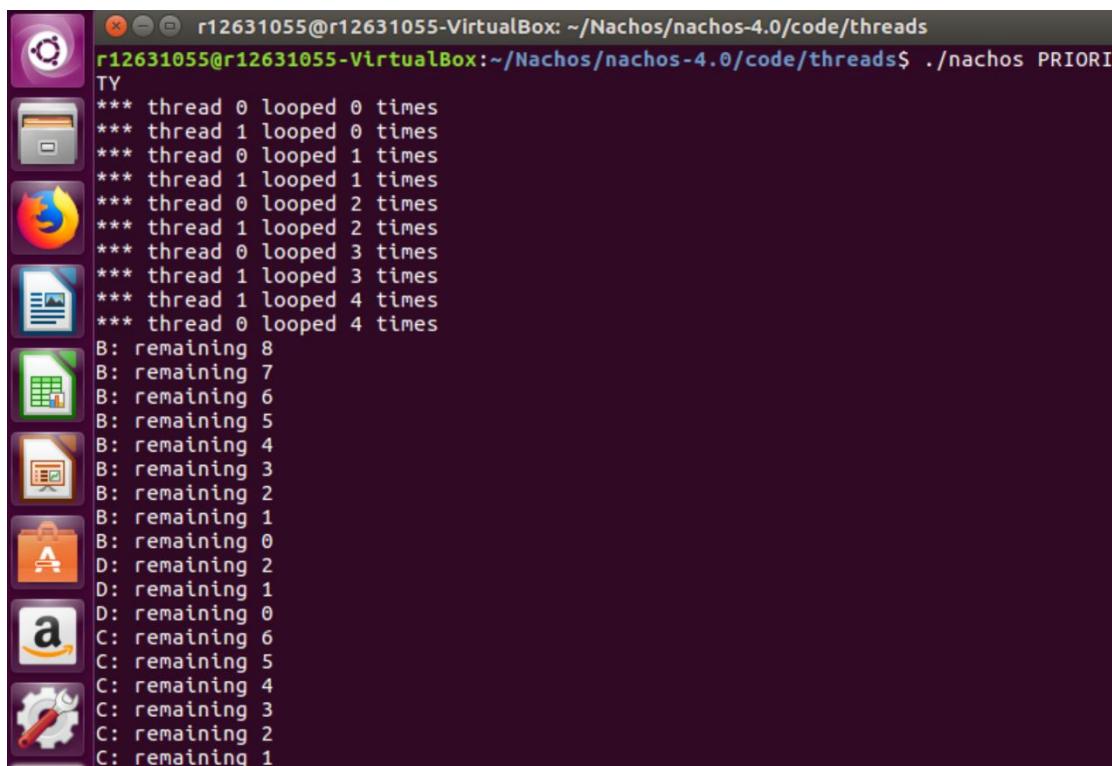
```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$ ./nachos SJF
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** 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
```



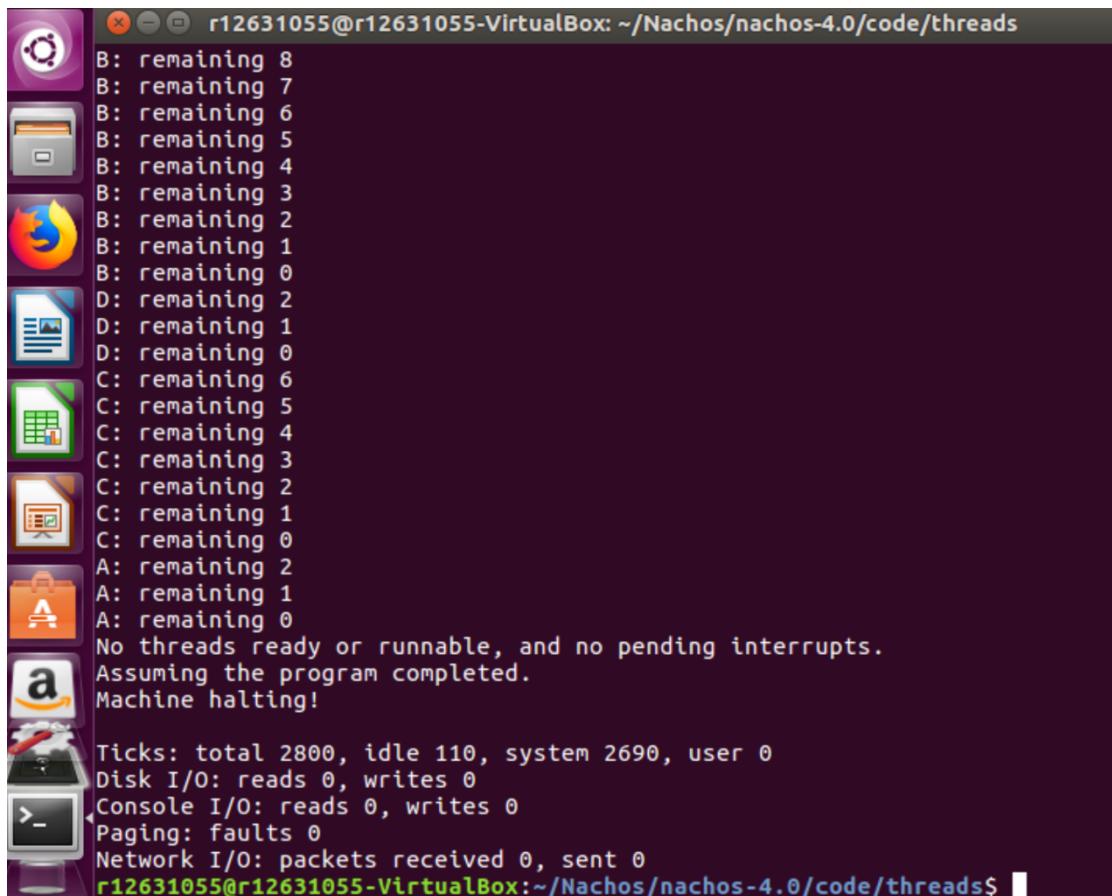
```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$ 
A: remaining 2
A: remaining 1
A: remaining 0
D: remaining 2
D: remaining 1
D: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
B: remaining 8
B: remaining 7
B: remaining 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
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$
```

從最簡短的 A 開始，再來是第二短的 D 之後 C 最後 B，由短至長順序處理。

再來是 Priority :



r12631055@r12631055-VirtualBox: ~/Nachos/nachos-4.0/code/threads
r12631055@r12631055-VirtualBox:~/Nachos/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
*** thread 1 looped 4 times
*** thread 0 looped 4 times
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
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



```
r12631055@r12631055-VirtualBox: ~/Nachos/nachos-4.0/code/threads  
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  
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  
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 2800, idle 110, system 2690, 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  
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$
```

會按照預先給定的重要次序來處理。

最後是 Round Robin

```
r12631055@r12631055-VirtualBox:~/Nachos/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
*** thread 1 looped 4 times
*** thread 0 looped 4 times
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
A: remaining 2
A: remaining 1
A: remaining 0
B: remaining 0
D: remaining 2
D: remaining 1

r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$ 
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
A: remaining 2
A: remaining 1
A: remaining 0
B: 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 2800, idle 120, system 2680, 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
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/threads$ 
```

設定 job 上限時間，像 B 因為超出時間，所以被中止後發配末端等待再執行。

Referrence:

https://jasonblog.github.io/note/gunmake/shen_ru_xue_xi_makeming_ling_he_mak_efile.html

<https://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/>

<https://blog.terrynini.tw/tw/OS-NachOS-HW1/>

https://hackmd.io/@Z_yUjsyqRzaD5rSUQ6JOVw/S1hiHr5D