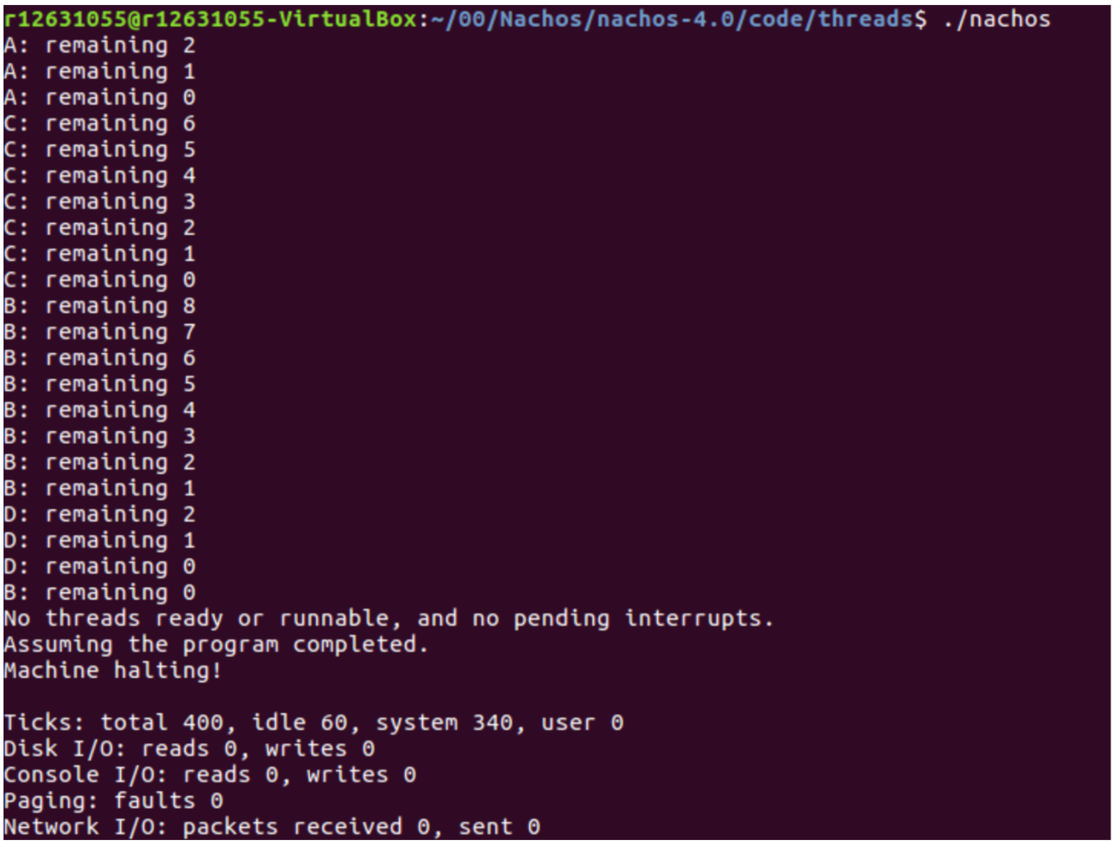
2023 FALL OS Project 3 Part2

R12631055 林東甫

Part2

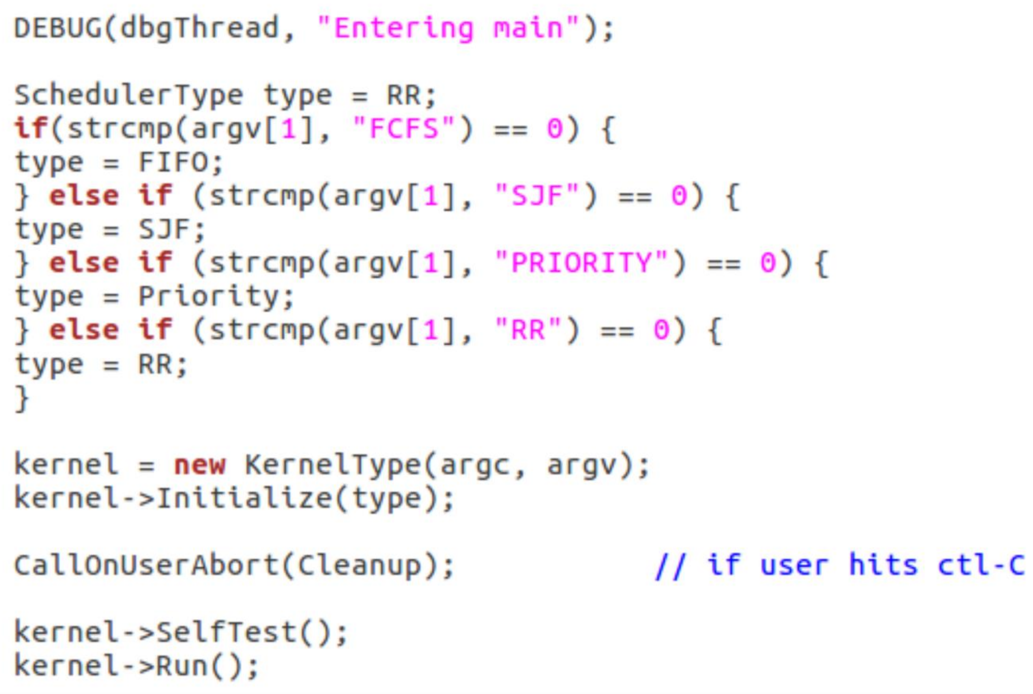
1. 問題是什麼？
    - **Round Robin 執行順序不符合預期，應該是 ABC 而不是 ACBDB。**



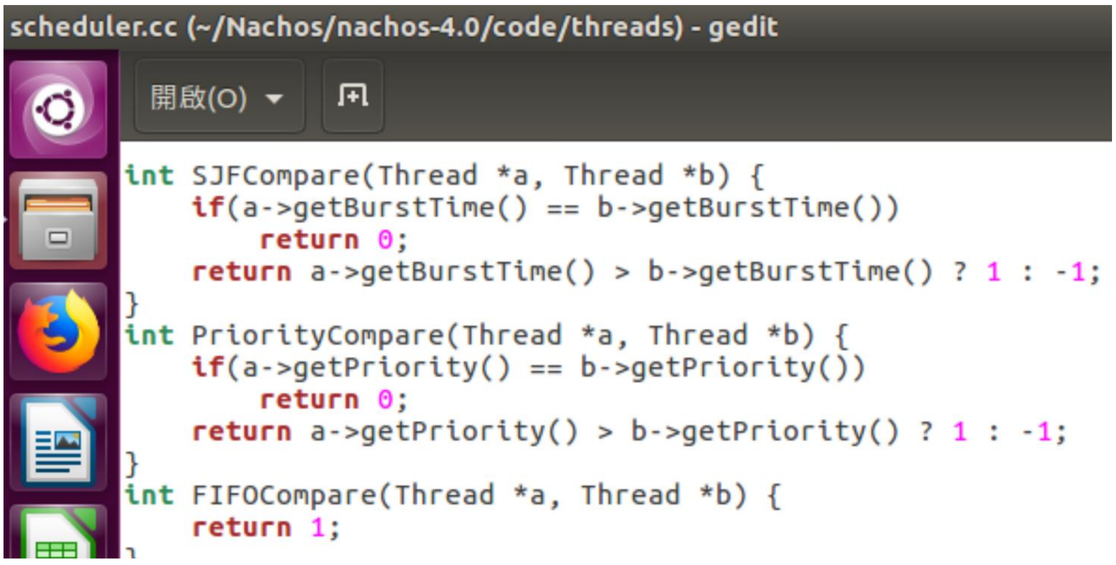
```
r12631055@r12631055-VirtualBox:~/OO/Nachos/nachos-4.0/code/threads$ ./nachos
A: remaining 2
A: remaining 1
A: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
D: remaining 2
D: remaining 1
D: remaining 0
B: remaining 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 400, idle 60, system 340, 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
```

2. 問題的原因為何？
    ○ 可能是由於 **Round-Robin** 算法的實現錯誤或 **Thread** 的優先級設定問題。

3. 找到這個問題的過程以及思考邏輯。
    ○ 首先，檢查 **Round-Robin** 調度算法的實現，確保它按照順序切換執行緒。

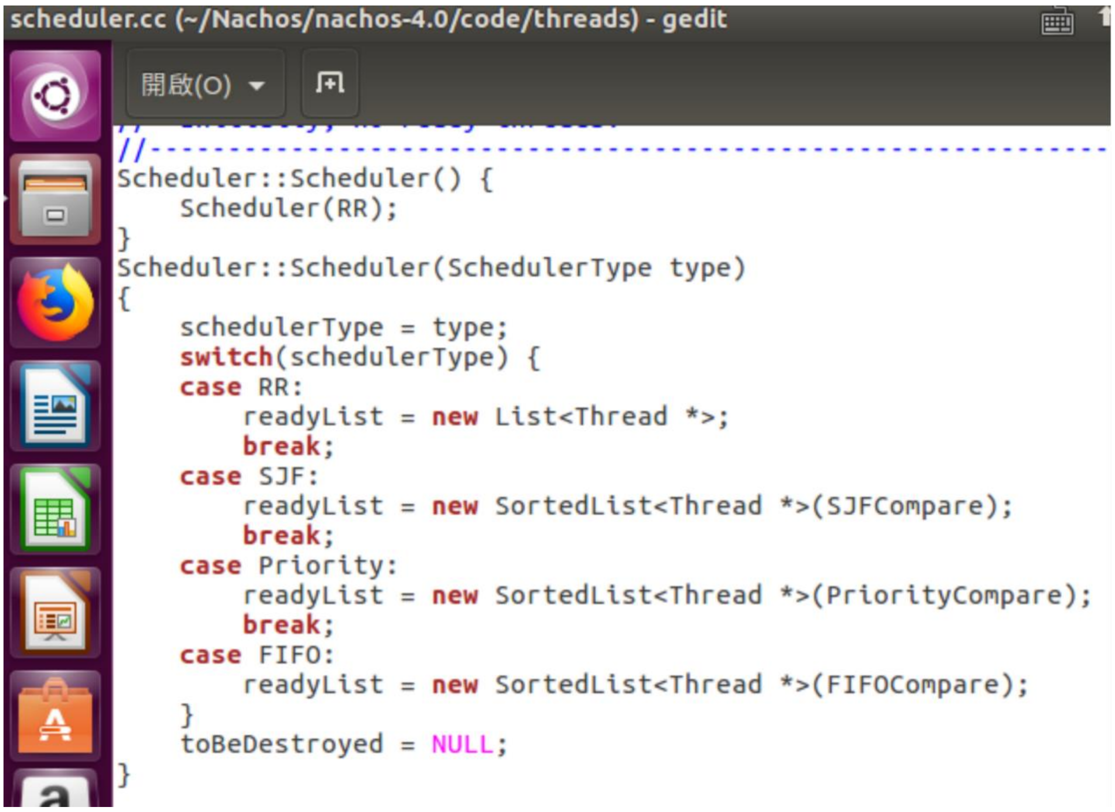


```
DEBUG(dbgThread, "Entering main");

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();
```



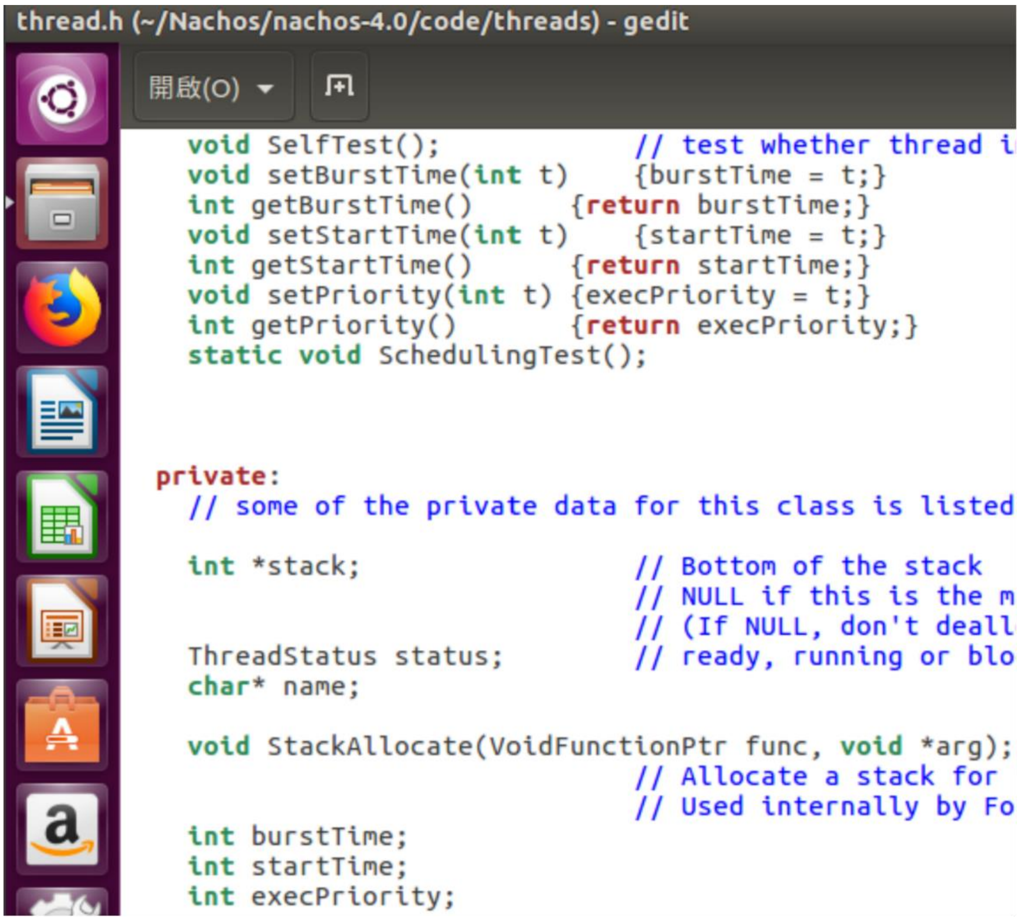
scheduler.cc (~/Nachos/nachos-4.0/code/threads) - gedit

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

```
//-----------------------------------------------------------------
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;
}
```

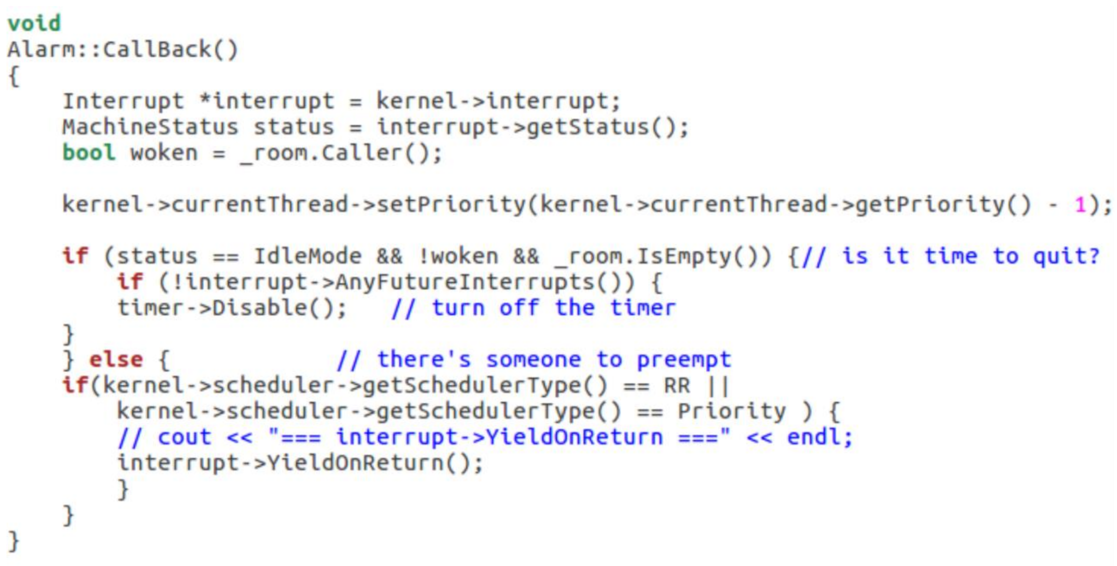
- 其次，檢查 **Thread** 的優先級，確保它們按照預期設定。


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

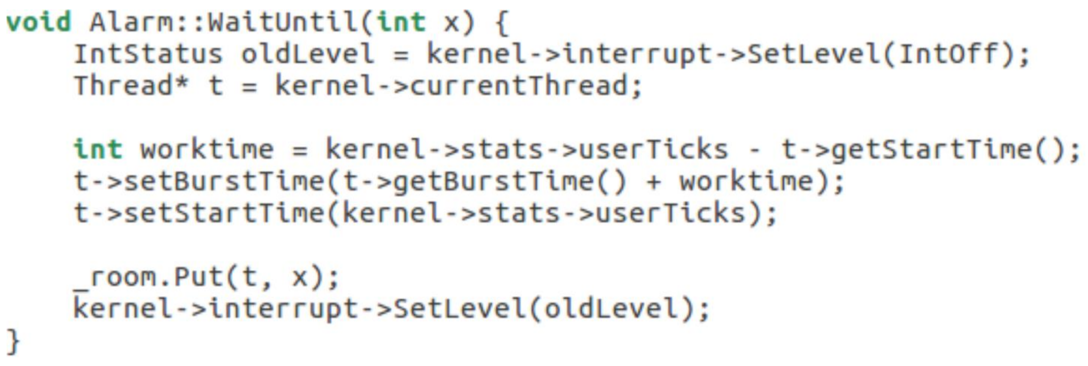
```
    void SelfTest();             // test whether thread i
    void setBurstTime(int t)     {burstTime = t;}
    int getBurstTime()       {return burstTime;}
    void setStartTime(int t)     {startTime = t;}
    int getStartTime()       {return startTime;}
    void setPriority(int t) {execPriority = 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 m
                                 // (If NULL, don't deall
    ThreadStatus status;         // ready, running or blo
    char* name;

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

- ○ 檢查是否有其他可能影響調度的因素。

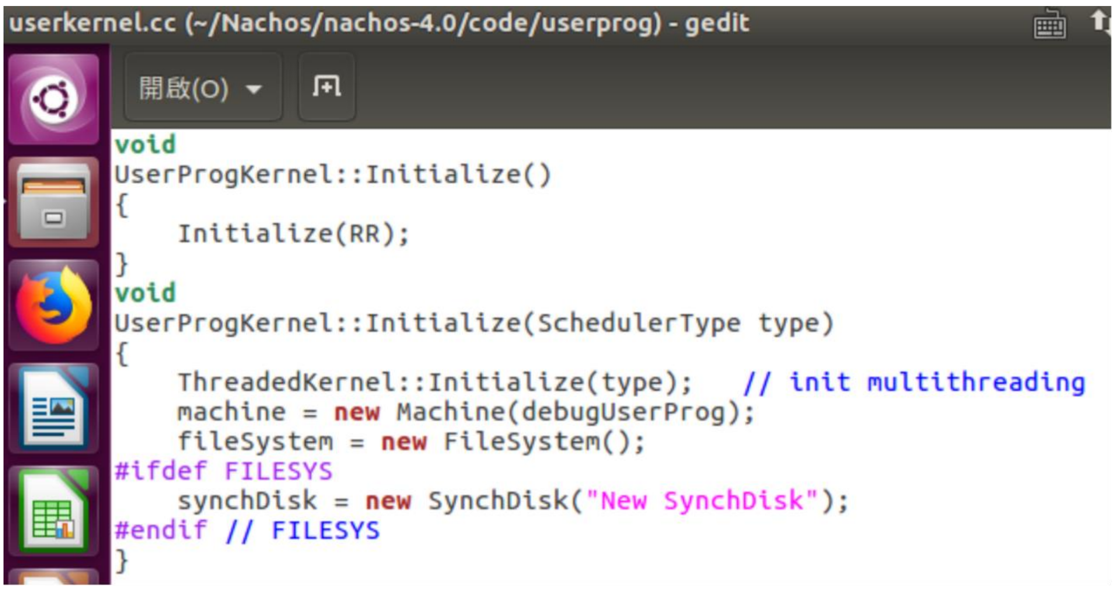
檢查 Alarm 和 userKernel



```
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();
        }
    }
}
```

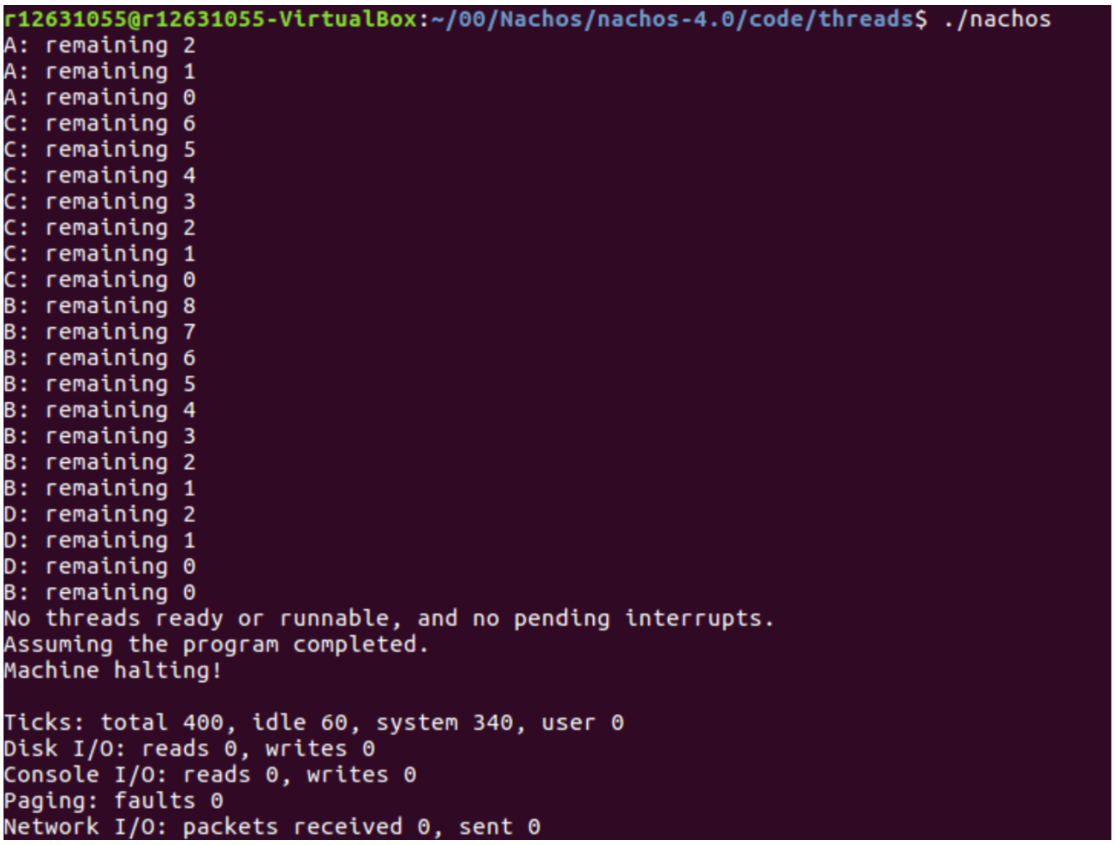


```
void Alarm::WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;

    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);
}
```



```
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
}
```

4. 要怎麼修改才能解決問題，請指出你修改的地方。
    - 檢查 **Round-Robin** 調度算法的實現，確保它按照順序切換執行緒。
    - 確保 **Thread** 的優先級按照預期設定。
5. 為什麼這樣修改後能解決問題。
    - 如果發現 **Round-Robin** 算法的實現有誤，修復它可以確保按照順序切換執行緒。
    - 如果 **Thread** 的優先級有誤，修正它可以確保按照預期設定。
6. 更多解決思路。
    - 檢查是否有其他影響調度的因素，例如是否有阻塞或等待的情況。
    - 考慮使用其他調度算法進行比較，確認問題是否僅存在於 **Round-Robin**。
    - 似乎其他的調度演算法都還能正常運作、也沒有特殊的錯誤情況。
    - 最後儘管做了許多檢查、修改跟測試，但看起來好像還是不太成功：



```
r12631055@r12631055-VirtualBox:~/00/Nachos/nachos-4.0/code/threads$ ./nachos
A: remaining 2
A: remaining 1
A: 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
D: remaining 2
D: remaining 1
D: remaining 0
B: remaining 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 400, idle 60, system 340, 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
```

7. 任何有幫助的說明與想法。
    - 如果在 **NachOS** 中使用優先級，確保遵從優先級的設定。
    - 檢查每個執行緒的狀態，例如是否有等待 **I/O** 或其他事件的情況，這可能會影響調度。
    - 對不起我盡力了 **orz**。

Referrence:

https://jasonblog.github.io/note/gunmake/shen_ru_xue_xi_makeming_ling_he_makefile.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/S1hiIHr5D