OS Project 2: System Call & CPU Scheduling

Student ID: R06921095

Name: Jui-Che Wu

➢ Motivation

✓ System Call

A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. To requests a service from the kernel of the operating system, System calls provide an essential interface between a process and the operating system. Thus, to accomplish our work easily, we may want to utilize the system call to tackle some specific job. Such as read(), write(), send(), open(), etc…. In this project, we are going to implement a "Sleep" system call.to halt the program for a before another execution.

✓ CPU Scheduling

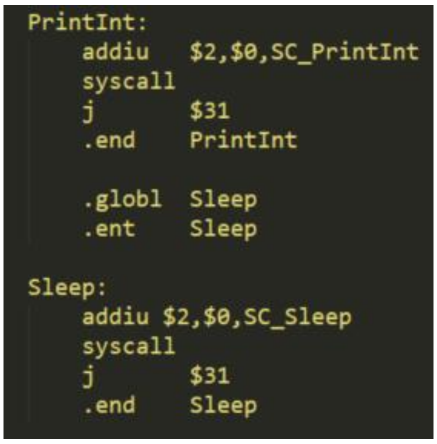
To obtain the maximum utilization of CPU and enhance the efficiency of multi-programming, a good CPU scheduling algorithm is required. The aim of CPU scheduling is to make the system efficient, fast and fair. Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

In this project, we're going to implement different scheduling algorithms such as First-Come-First-Service (FCFS), Shortest-Job-First (SJF), Priority, and the default Round-Robin (RR) algorithms.

➢ Implementation

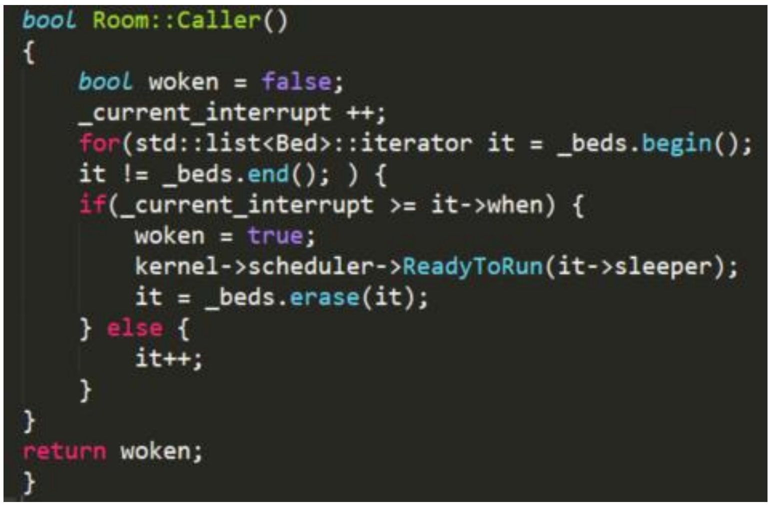
✓ System Call

In the previous project, we've implemented a system call called "PrintInt". Then, we can implement the "Sleep" system call by using the similar method. First, we define the Sleep() function in the syscall.h for other program to utilize this sleep functionality. Next, as we know, nachos is using a MIPS ISA, which means that the program will be compiled to MIPS binary code after compile. Thus, we have to use the MIPS assembling language to implement the system call "Sleep" as shown in the following figure. We can implement it as the similar way as "PrintInt".



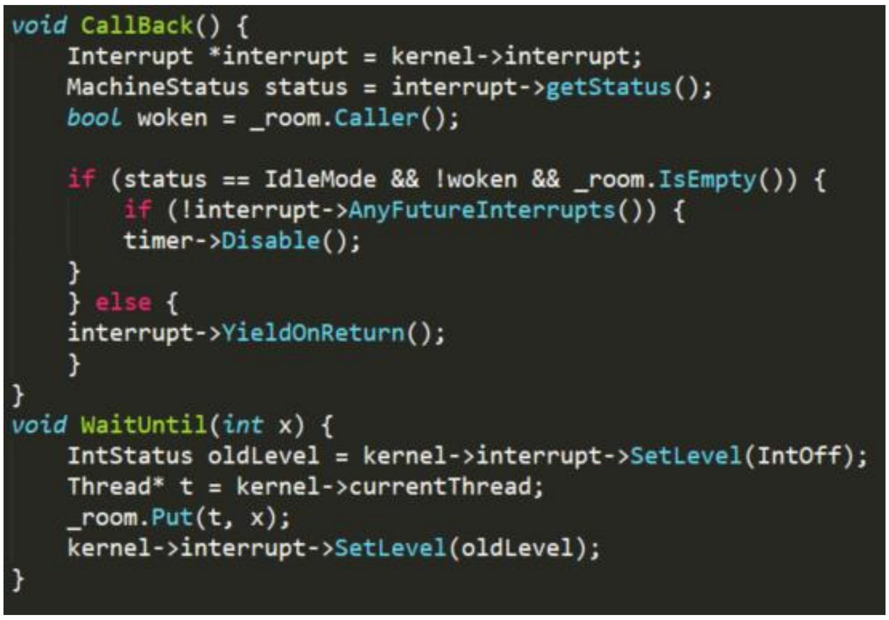
```
PrintInt:
    addiu   $2,$0,SC_PrintInt
    syscall
    j       $31
    .end    PrintInt

    .globl  Sleep
    .ent    Sleep

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

To calculate the time for the program to wake up. We will need to implement a counter to calculate the time. we use a integer to record the time of current interrupt, then we can obtain the global time, and evoke the wakeup function after the given sleep time. The time of waking up is storing in a list, every time the caller is called, the list will be checked whether the time is smaller than the current interrupt.



```
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;
}
```

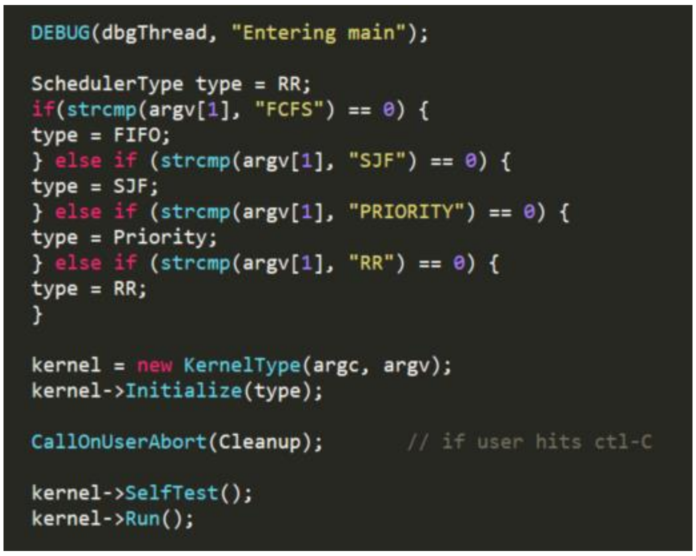
Then, we will implement a waituntil function and a callback function., these function will handle the main part of the sleep system call. The waituntil function will be called when a thread going to sleep, and the callback function will check which thread should wake up, which is an interrupt handle for the timer device. This routine is called each time there is a time interrupt. And it will check if it is the time to quit.



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

    if (status == IdleMode && !woken && _room.IsEmpty()) {
        if (!interrupt->AnyFutureInterrupts()) {
        timer->Disable();
    }
    } else {
    interrupt->YieldOnReturn();
    }
}
void WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;
    _room.Put(t, x);
    kernel->interrupt->SetLevel(oldLevel);
}
```

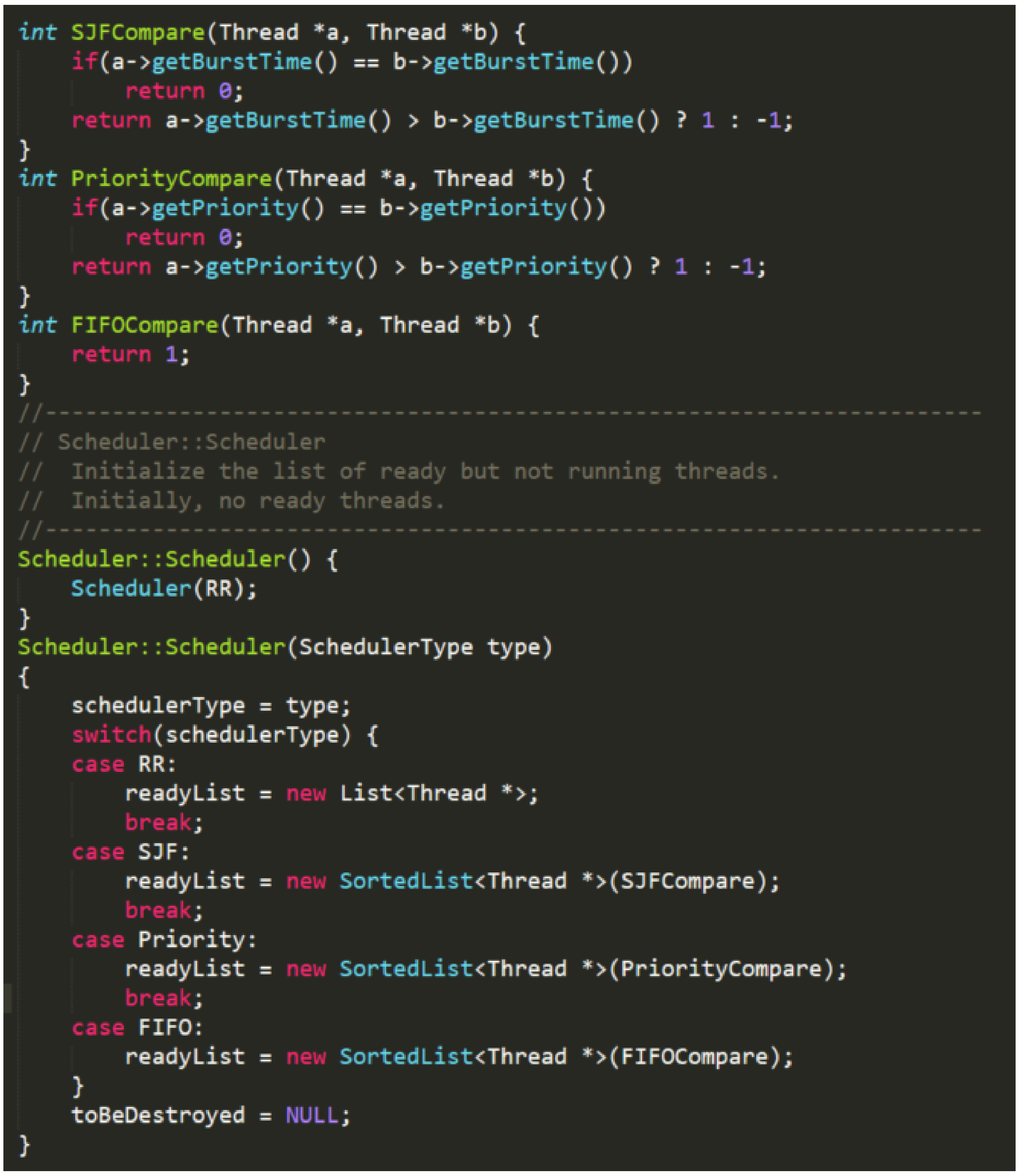
✓ CPU Scheduling

First, we write a interface to use the corresponding algorithm in main.cc. The following code is to parse the string of the given argument vector, and the parsing argument is compared with the given algorithm and chose it as our scheduling algorithm.



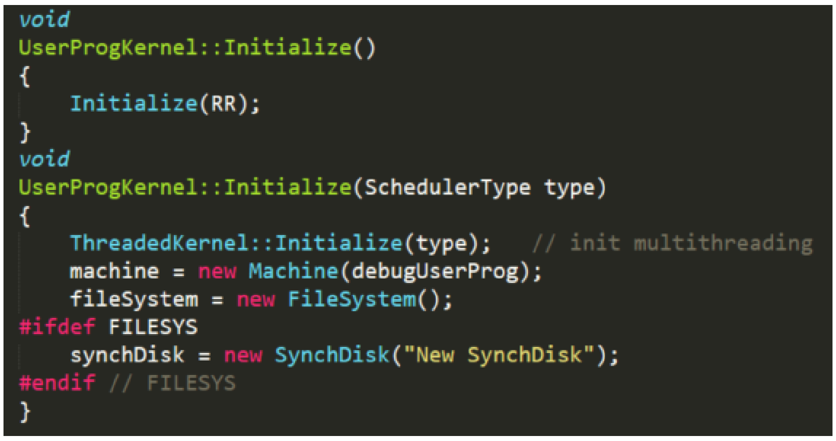
```
DEBUG(dbgThread, "Entering main");

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();
```

After parsing the command, we will use the concept of polymorphism to prepare the corresponding ready list for scheduling and declare the corresponding compare function as shown in the following figure. By default, we will use the Round Robin (RR) algorithm, which is declared in the default constructor of the scheduler.

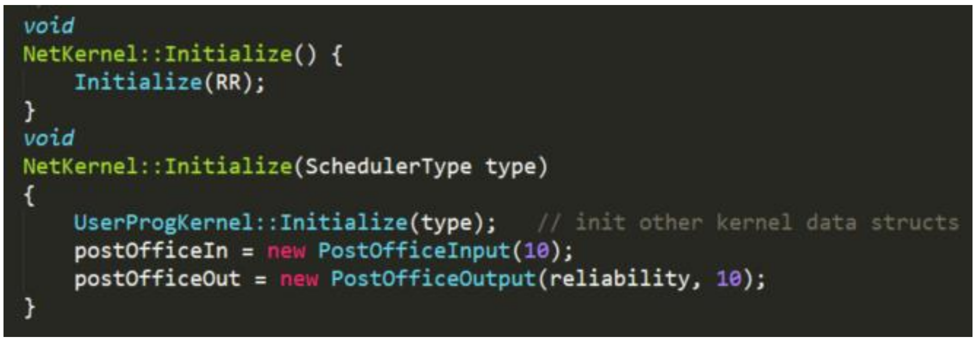


```
int SJFCompare(Thread *a, Thread *b) {
    if(a->getBurstTime() == b->getBurstTime())
        return 0;
    return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
}
int PriorityCompare(Thread *a, Thread *b) {
    if(a->getPriority() == b->getPriority())
        return 0;
    return a->getPriority()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

Finally, we will also need to declare and implement the initialize function in both userkernel and netkernel. Otherwise, it will result in compile error.



```
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
}
```

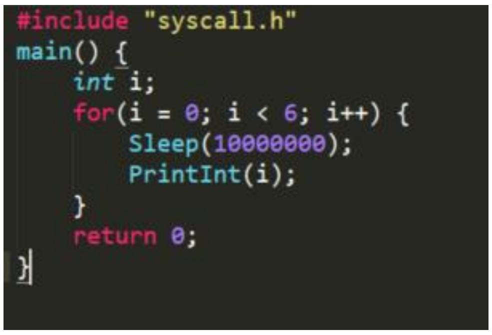


```
void
NetKernel::Initialize() {
    Initialize(RR);
}
void
NetKernel::Initialize(SchedulerType type)
{
    UserProgKernel::Initialize(type);   // init other kernel data structs
    postOfficeIn = new PostOfficeInput(10);
    postOfficeOut = new PostOfficeOutput(reliability, 10);
}
```

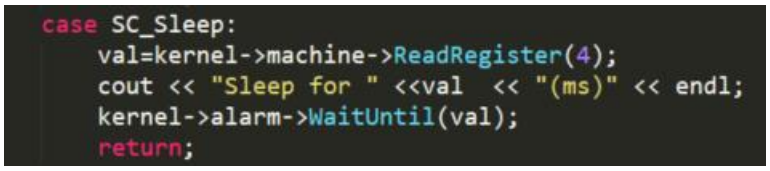
➢ Result

✓ System Call

In this part, we will test the Sleep function by using a test case. Similar in the previous project, we will print the integer from 0 to 5 using a for loop. And in each loop, a sleep function will be called. Which is evoked from the exceptionhandler. And printout the sleep time in each loop as well.

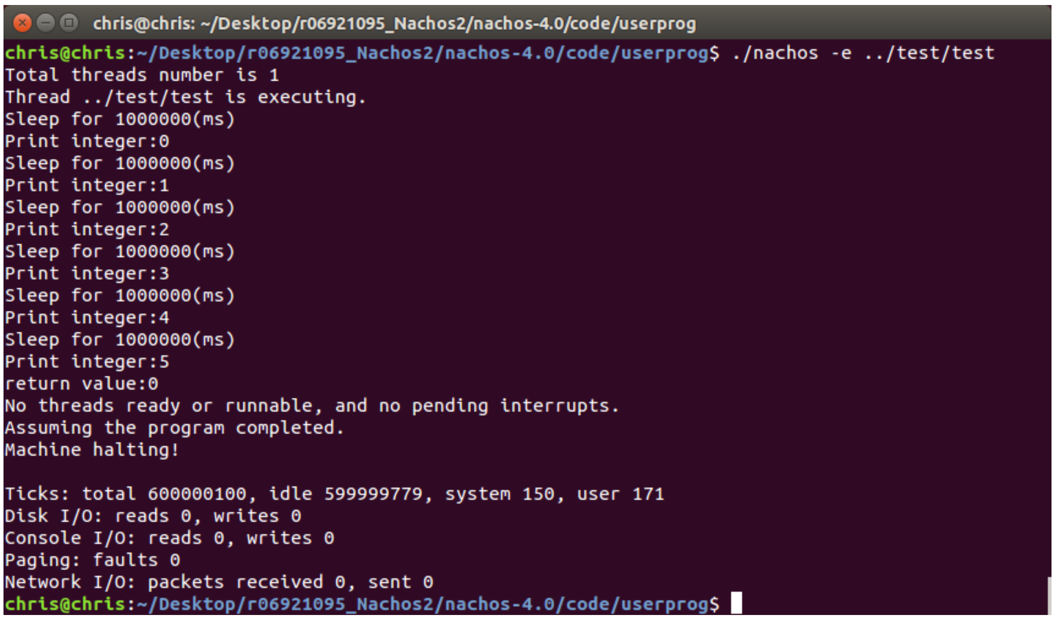


```
#include "syscall.h"
main() {
    int i;
    for(i = 0; i < 6; i++) {
        Sleep(10000000);
        PrintInt(i);
    }
    return 0;
}
```



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

The following figure is the output of the testing result. The absolute time may not be accurate compare to the given time in Sleep function. However, the relative time is roughly correct.

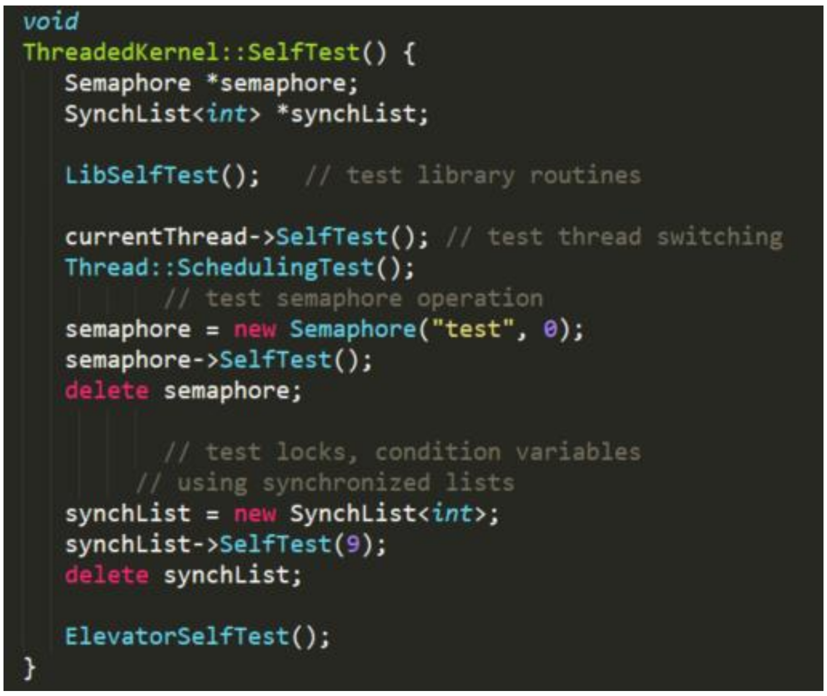


```
chris@chris: ~/Desktop/r06921095_Nachos2/nachos-4.0/code/userprog
chris@chris:~/Desktop/r06921095_Nachos2/nachos-4.0/code/userprog$ ./nachos -e ../test/test
Total threads number is 1
Thread ../test/test is executing.
Sleep for 1000000(ms)
Print integer:0
Sleep for 1000000(ms)
Print integer:1
Sleep 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
chris@chris:~/Desktop/r06921095_Nachos2/nachos-4.0/code/userprog$
```

✓ CPU Scheduling

To test our scheduling algorithm, we can utilize the given selftest function in many class, what we have to do is to write our test code in the selftest function until it is called. This can test whether our scheduling order of the corresponding algorithm is correct or not. Finally, we can utilize the following command to execute out algorithm in the directory of the

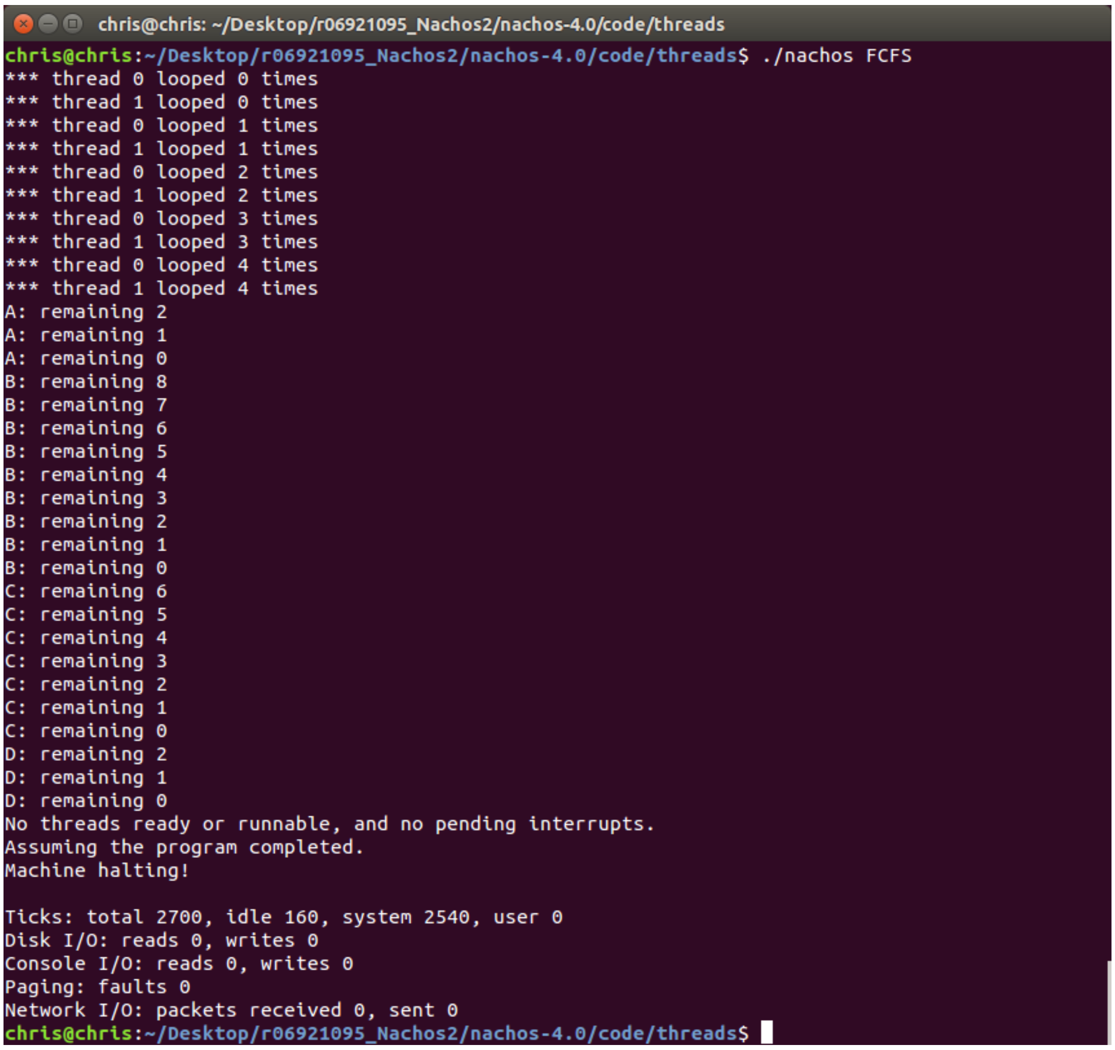
```
cd nachos/code/thread
/nachos FCFS
/nachos SJF
/nachos PRIORITY
/nachos RR (default algorithm)
```



```
void
ThreadedKernel::SelfTest() {
    Semaphore *semaphore;
    SynchList<int> *synchList;

    LibSelfTest();    // test library routines

    currentThread->SelfTest(); // test thread switching
    Thread::SchedulingTest();
                // test semaphore operation
    semaphore = new Semaphore("test", 0);
    semaphore->SelfTest();
    delete semaphore;

                // test locks, condition variables
              // using synchronized lists
    synchList = new SynchList<int>;
    synchList->SelfTest(9);
    delete synchList;

    ElevatorSelfTest();
}
```

Finally, we can obtain the result of the corresponding algorithm.

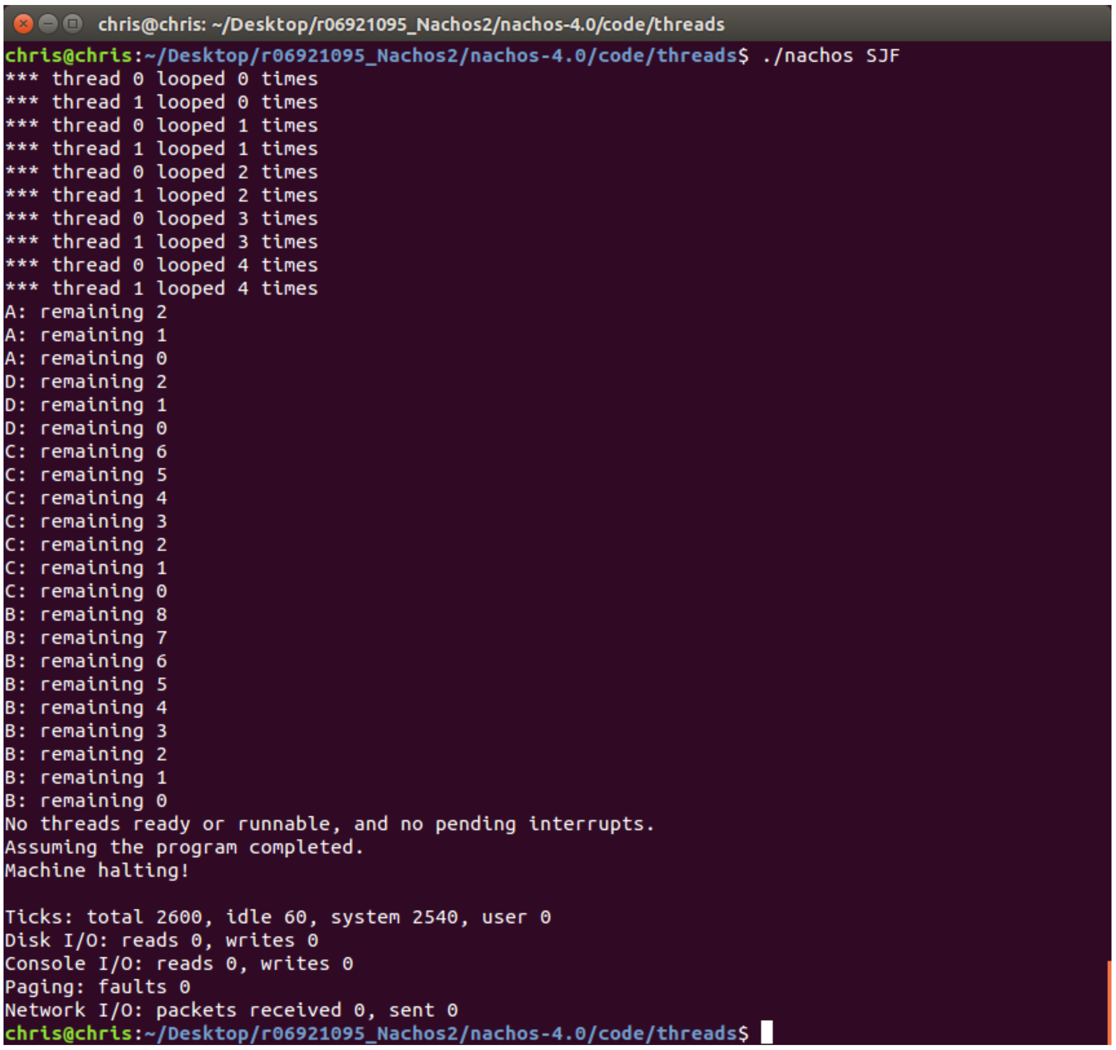
- First-Come-First-Service (FCFS)



chris@chris: ~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads

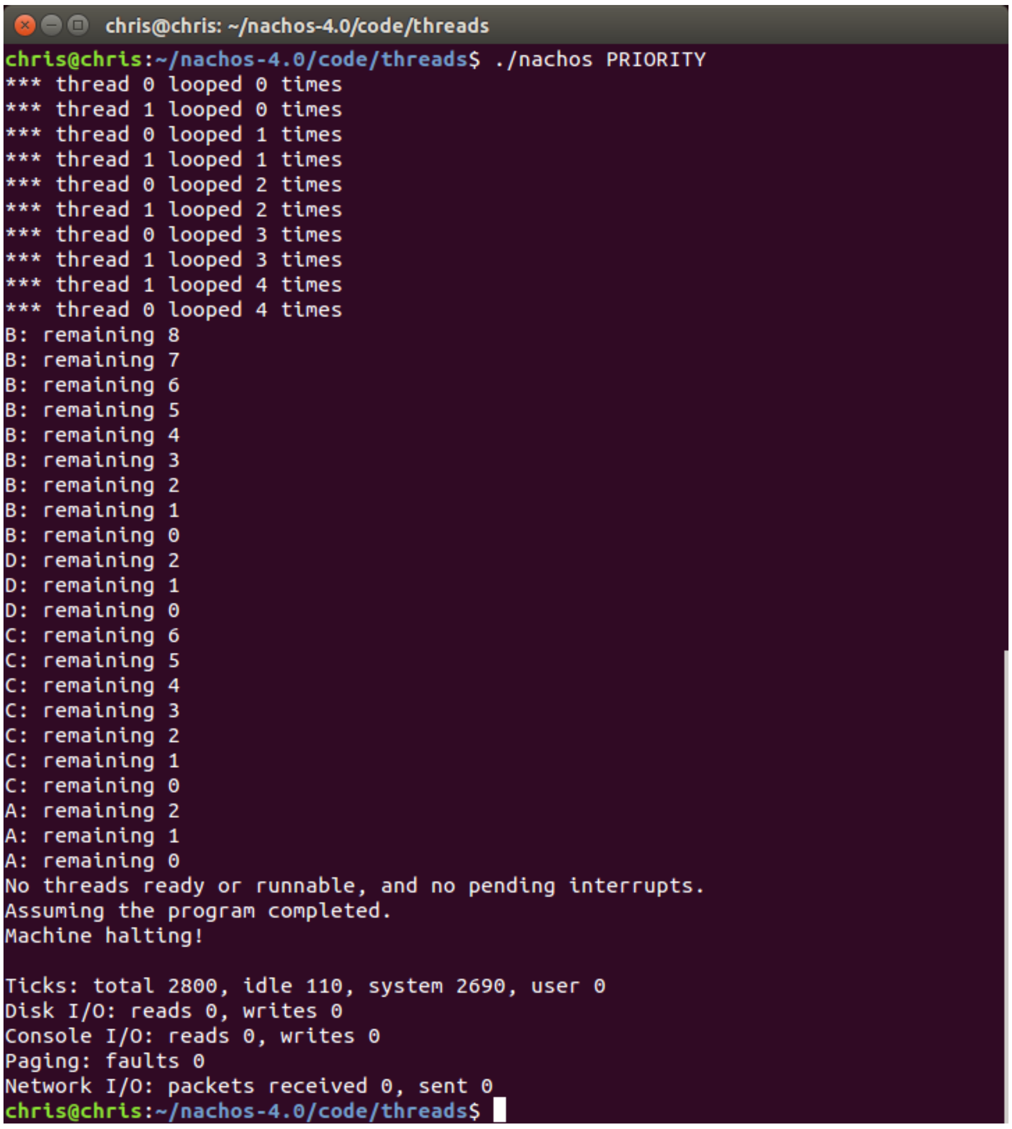
```
chris@chris:~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads$ ./nachos FCFS
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 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
chris@chris:~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads$
```

- Shortest-Job-First (SJF)



```
chris@chris: ~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads
chris@chris:~/Desktop/r06921095_Nachos2/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
chris@chris:~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads$
```

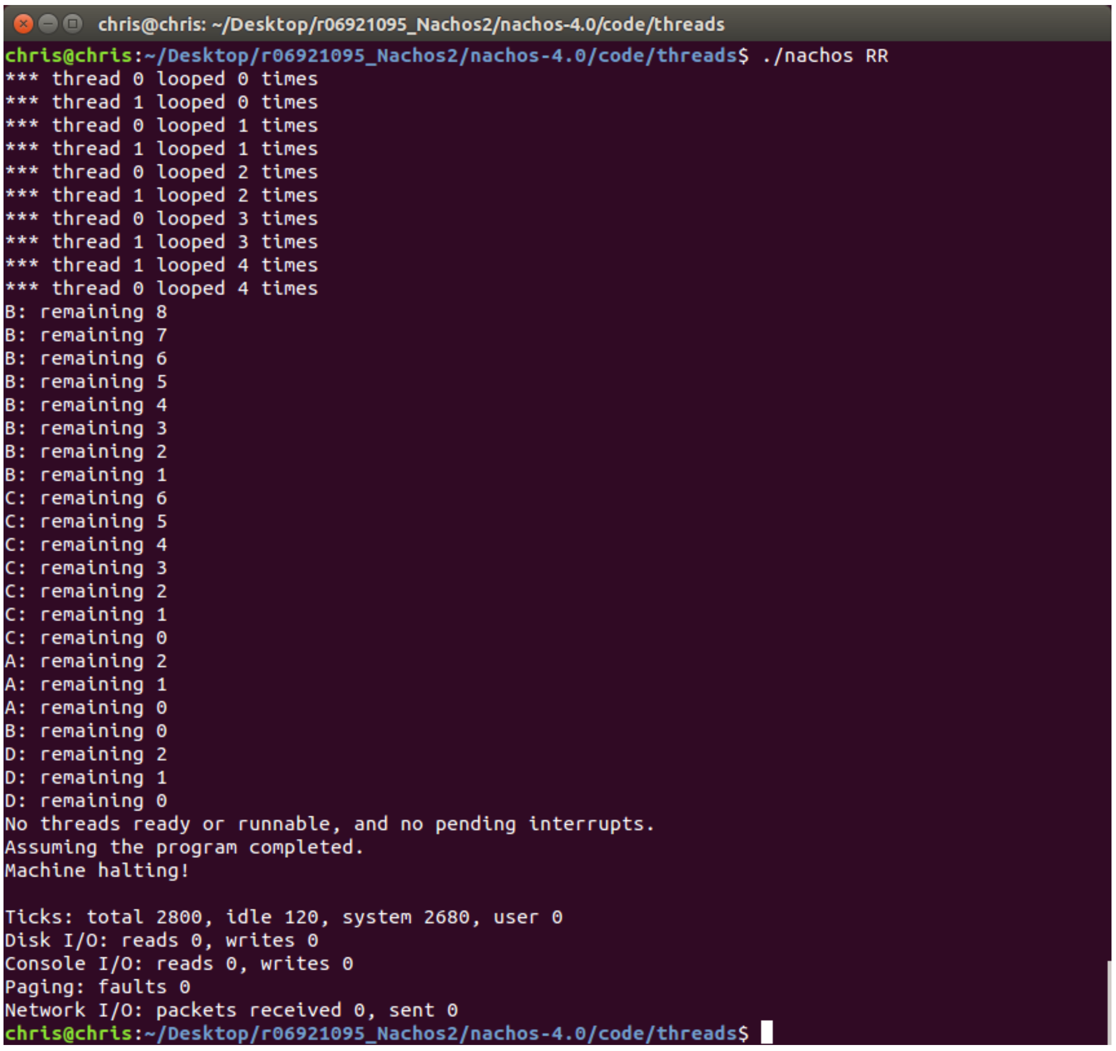
- Priority



chris@chris: ~/nachos-4.0/code/threads

```
chris@chris:~/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
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
chris@chris:~/nachos-4.0/code/threads$
```

- Round-Robin (default)



chris@chris: ~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads

```
chris@chris:~/Desktop/r06921095_Nachos2/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
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
chris@chris:~/Desktop/r06921095_Nachos2/nachos-4.0/code/threads$
```