Project 2

工科海洋四 劉名凱

1. Sleep

Motivation:

In this problem we need to implement Sleep system call. This involving not only writing c++ code but also require some machine language. Since I have never implemented a system call before, I started by reading the code of syscall.h. Than according to guidance in the pdf, alarm.h is the key to this problem, so I also read the code in alarm.h and alarm.cc. Finally, I try to make thread sleep by calling function in alarm.h.

Implementation:

First, I add a declaration of Sleep system call in syscall.h by following the declaration already existed in the code.

```
21  #define SC_Halt      0
22  #define SC_Exit      1
23  #define SC_Exec      2
24  #define SC_Join      3
25  #define SC_Create    4
26  #define SC_Open      5
27  #define SC_Read      6
28  #define SC_Write     7
29  #define SC_Close     8
30  #define SC_ThreadFork   9
31  #define SC_ThreadYield  10
32  #define SC_PrintInt 11
33  #define SC_Sleep     12
```

```
129  void ThreadYield();
130
131  void PrintInt(int number);  //my System Call
132  void Sleep(int number);     //Sleep System call
```

Then adding assembly language in start.s, making Sleep() available to Nachos kernel and letting Nachos kernel know which position in the register contains the function parameter. (I have never write assembly language before, so I google for help

start.s

```
135 PrintInt:
136     addiu   $2,$0,SC_PrintInt
137     syscall
138     j       $31
139     .end    PrintInt
140
141     .globl  Sleep
142     .ent    Sleep
143 Sleep:
144     addiu   $2,$0,SC_Sleep
145     syscall
146     j   $31
147     .end    Sleep
```

By adding Sleep case in exception.cc, kernel will know what to do when processes call Sleep(). When Sleep() is called, system will print out the sleep time and call WaitUntil() to force processes fall into sleep.

excepion.cc

```
switch (which) {
case SyscallException:
    switch(type) {
    case SC_Halt:
        DEBUG(dbgAddr, "Shutdown, initiated by user program.\n");
        kernel->interrupt->Halt();
        break;
    case SC_PrintInt:
        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);
        return;
```

Define a class sleepList in alarm.h to store threads that were put into sleep.

PutToSleep(): put threads into sleep.

IsEmpty(): determine whether the sleepList is empty.

_threadlist: tracks threads which were put into sleep.

Class sleepThread: record the sleeping state of a thread

alarm.h

```
27  class sleepList{
28    public:
29      sleepList():_current_interrupt(0) {};
30
31      void PutToSleep(Thread* t, int x);
32
33    bool PutToReady();
34
35    bool IsEmpty();
36
37    private:
38      class sleepThread{
39        public:
40          sleepThread(Thread* t, int x):
41            sleeper(t), when(x) {};
42
43    Thread* sleeper;
44    int when;
45      };
46    int _current_interrupt;
47    std::list<sleepThread> _threadlist;
48  };
49
```

Add a sleepList variable in class Alarm, so that each process' alarm can track which thread is put into sleep.

alarm.h

```
50  // The following class defines a software alarm clock.
51  class Alarm : public CallBackObj {
52    public:
53      Alarm(bool doRandomYield);  // Initialize the timer, and callback
54          // to "toCall" every time slice.
55      ~Alarm() { delete timer; }
56
57      void WaitUntil(int x);  // suspend execution until time > now + x
58
59    private:
60      Timer *timer;   // the hardware timer device
61      sleepList _sleepList;
62      void CallBack();    // called when the hardware
63          // timer generates an interrupt
64  };
65
66  #endif // ALARM_H
```

Then explicit define functions in alarm.cc

CallBack:

Add constrains in if statement to make sure when the alarm is tuned off, there are no sleeping threads.

alarm.cc

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

Empty():

Check is _threadlist is empty.

alarm.cc

```
76  bool
77  sleepList::IsEmpty()
78  {
79      return _threadlist.size() == 0;
80  }
```

WaitUntil():

Put a thread into sleep. When putting threads into sleep, we have to turn off interrupt until target thread is successfully put into sleep

alarm.cc

Turn off interrupt

Turn on interrupt

```
65  void
66  Alarm::WaitUntil(int x)
67  {
68      IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
69      Thread* t = kernel->currentThread;
70      cout << "Alarm::WaitUntil go sleep" << endl;
71      _sleepList.PutToSleep(t, x);
72
73      kernel->interrupt->SetLevel(oldLevel);
74  }
```

PutToReady():

When a thread has slept for target time (_current_interrupt >= when), wake up the thread.

alarm.cc

```
bool
sleepList::PutToReady() {
    bool woken = false;
    _current_interrupt ++;
    for(std::list<sleepThread>::iterator it = _threadlist.begin(); it != _threadlist.end();)
    {
        if(_current_interrupt >= it->when) {
            woken = true;
            cout << "sleepList::PutToReady Thread woken" << endl;
            kernel->scheduler->ReadyToRun(it->sleeper);
            it = _threadlist.erase(it);
        } else {
            it++;
        }
    }
    return woken;
}
```

Finally, write sleep.c and sleep2.c. Period in sleep.c is ten times larger than those in sleep2.c. Therefore, when executing both process simultaneously, terminal should show 1 every ten 2 are printed.

sleep.c

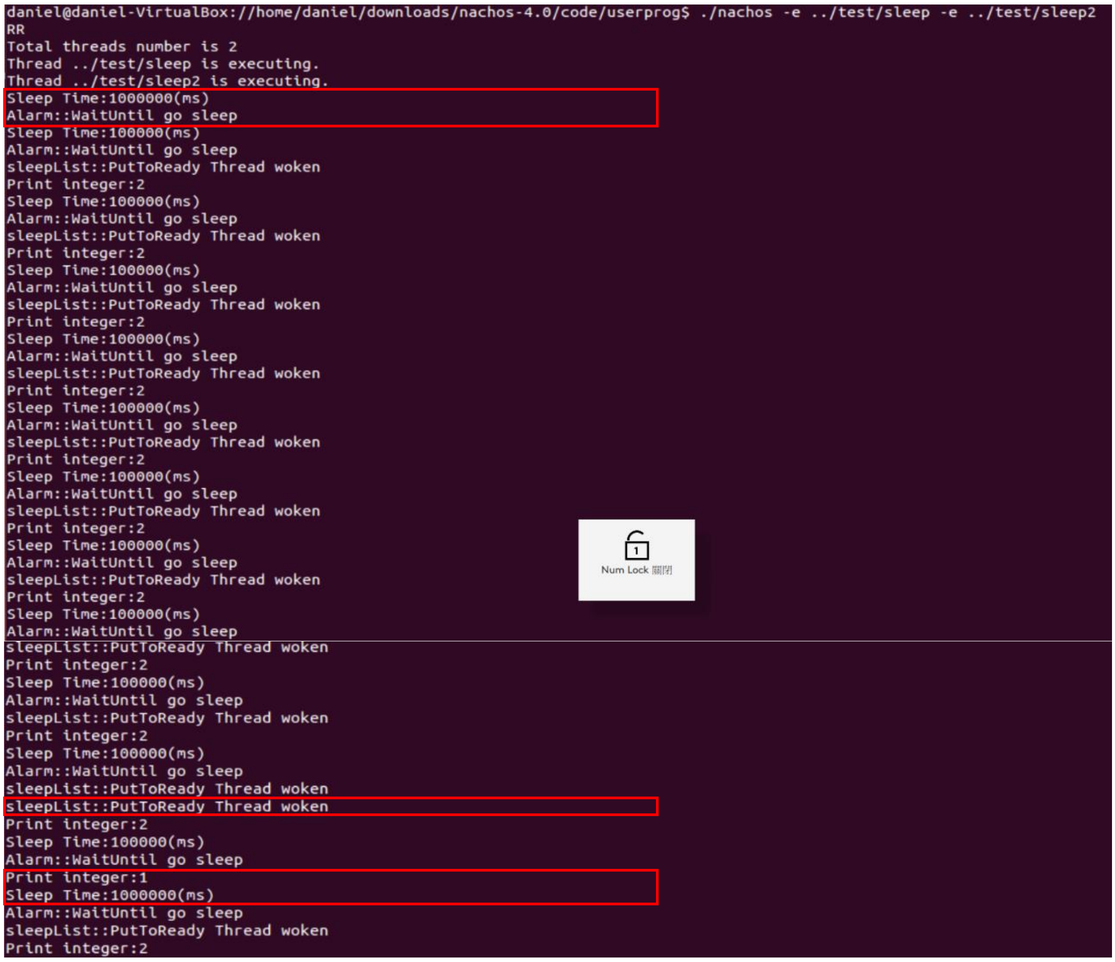
```
#include "syscall.h"

main() {
    int i;
    for(i = 0; i<5; i++){
        Sleep(1000000);
        PrintInt(1);
    }
    return 0;
}
```

sleep2.c

```
#include "syscall.h"

main() {
    int i;
    for(i = 0; i < 20; i++) {
        Sleep(100000);
        PrintInt(2);
    }
    return 0;
}
```



2. CPU Scheduling

Motivation:

Since CPU scheduling is simply a list containing the execution sequence, I accomplish this part quicker than problem1. RR, FCFS, SJF, Priority are not difficult to understand, just implement corresponding scheduling list.

Implementation:

Firstly, modify SelfTest() and SimpleThread() in thread.cc so that we can perform a simple test with our CPU scheduling.

SelfTest():

Initialize four thread A, B, C and D, each has its own priority and burst time. After initializing threads, fork a child thread and perform SimpleThread().

thread.cc

```
430  void
431  Thread::SelfTest()
432  {
433      DEBUG(dbgThread, "Entering Thread::SelfTest");
434
435      const int number     = 4;
436      char *name[number]   = {"A", "B", "C", "D"};
437      int burst[number]    = {3, 10, 4, 9};
438      int priority[number] = {4, 5, 3, 1};
439
440
441      Thread *t;
442      for (int i = 0; i < number; i ++) {
443          t = new Thread(name[i]);
444          t->setPriority(priority[i]);
445          t->setBurstTime(burst[i]);
446          t->Fork((VoidFunctionPtr) SimpleThread, (void *)NULL);
447      }
448      kernel->currentThread->Yield();
449  }
```

SimpleThread():

Subtract current thread's burst time by one and print out its name and remaining burst time.

thread.cc

```
412  static void
413  SimpleThread()
414  {
415      Thread *thread = kernel->currentThread;
416      while (thread->getBurstTime() > 0) {
417          thread->setBurstTime(thread->getBurstTime() - 1);
418          printf("%s: %d\n", kernel->currentThread->getName(),
419                  kernel->currentThread->getBurstTime());
420          kernel->interrupt->OneTick();
421      }
422  }
```

Also add function prototype in thread.h

thread.h

```
106     char* getName() { return (name); }
107     void Print() { cout << name; }
108     void SelfTest();    // test whether thread impl is working
109
110     // asdasd
111     void setBurstTime(int t)  {burstTime = t;}
112     int getBurstTime()    {return burstTime;}
113     void setPriority(int t) {priority = t;}
114     int getPriority()   {return priority;}
115
116   private:
117     // some of the private data for this class is listed above
118
119     int *stack;     // Bottom of the stack
120         // NULL if this is the main thread
121         // (If NULL, don't deallocate stack)
122     ThreadStatus status;  // ready, running or blocked
123     char* name;
124
125     //asdasd
126     int burstTime;
127     int priority;
```

In kernel.cc modify ThreadedKernel's construction function. When establishing a new thread, we can also feed a argument to assign a scheduler method. (Default is RoundRobin)

kernel.cc

```
25  ThreadedKernel::ThreadedKernel(int argc, char **argv)
26  {
27      randomSlice = FALSE;
28      type = RR;
29      for (int i = 1; i < argc; i++) {
30          if (strcmp(argv[i], "-rs") == 0) {
31              ASSERT(i + 1 < argc);
32              RandomInit(atoi(argv[i + 1]));// initialize pseudo-random
33                          // number generator
34              randomSlice = TRUE;
35              i++;
36          }
37          else if (strcmp(argv[i], "-u") == 0) {
38              cout << "Partial usage: nachos [-rs randomSeed]\n";
39          }
40          else if(strcmp(argv[i], "RR") == 0) {
41              type = RR;
42          }
43          else if (strcmp(argv[i], "FCFS") == 0) {
44              type = FIFO;
45          }
46          else if (strcmp(argv[i], "PRIORITY") == 0) {
47              type = Priority;
48          }
49          else if (strcmp(argv[i], "SJF") == 0) {
50              type = SJF;
51          }
52      }
53  }
```

Also pass target scheduler type to Scheduler variable in ThreadedKernel::Initilize().

```
68      scheduler = new Scheduler(type);    // initialize the ready queue
69      if(scheduler->getSchedulerType() == RR){
70          alarm = new Alarm(randomSlice); // start up time slicing
71      }
```

kernel.cc

Add variable in class ThreadedKernel to determine target scheduler type.

```
46      bool randomSlice;   // enable pseudo-random time slicing
47      SchedulerType type;
```

kernel.cc

Add Scheduler construction function that can read scheduler type and add functions that can set and get scheduler type.

```
27  class Scheduler {
28    public:
29    Scheduler();     // Initialize list of ready threads zz
30    Scheduler(SchedulerType type);
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    void setSchedulerType(SchedulerType t) {schedulerType = t;}
45    SchedulerType getSchedulerType() {return schedulerType;}
46    private:
47    SchedulerType schedulerType;
48    List<Thread *> *readyList;  // queue of threads that are ready to run,
49                        // but not running
50    Thread *toBeDestroyed;    // finishing thread to be destroyed
51                        // by the next thread that runs
52  };
```

scheduler.h

Define different types of list comparators for each scheduling method in scheduler.cc. (Round Robin basically same as FCFS but deal with threads alternatively. Moreover, it's default scheduler method in nachos)

scheduler.cc

```
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;
}

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

Scheduler construction. Default is Round Robin method. When specifying a scheduler type, initialize an execution list with its corresponding list comparator.

scheduler.cc

```
47  Scheduler::Scheduler()
48  {
49      Scheduler(RR);
50  }
51
52
53  Scheduler::Scheduler(SchedulerType type)
54  {
55      schedulerType = type;
56      switch(schedulerType) {
57          case RR:
58              cout<<"RR"<<endl;
59              readyList = new List<Thread *>;
60              break;
61          case SJF:
62              cout<<"SJF"<<endl;
63              readyList = new SortedList<Thread *>(SJFCompare);
64              break;
65          case Priority:
66              cout<<"Priority"<<endl;
67              readyList = new SortedList<Thread *>(PriorityCompare);
68              break;
69          case FIFO:
70              cout<<"FIFO"<<endl;
71              readyList = new SortedList<Thread *>(FIFOCompare);
72              break;
73      }
74      toBeDestroyed = NULL;
75  }
```

Test with SelfTest() in thread.cc

RR

```
daniel@daniel-VirtualBox://home/daniel/downloads/nachos-4.0/code/threads$ ./nachos RR
RR
A: 2
A: 1
A: 0
C: 3
C: 2
C: 1
C: 0
D: 8
D: 7
D: 6
D: 5
B: 9
B: 8
B: 7
B: 6
B: 5
B: 4
D: 4
D: 3
D: 2
D: 1
D: 0
B: 3
B: 2
B: 1
B: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

FCFS

```
daniel@daniel-VirtualBox://home/daniel/downloads/nachos-4.0/code/threads$ ./nachos FCFS
FIFO
A: 2
A: 1
A: 0
B: 9
B: 8
B: 7
B: 6
B: 5
B: 4
B: 3
B: 2
B: 1
B: 0
C: 3
C: 2
C: 1
C: 0
D: 8
D: 7
D: 6
D: 5
D: 4
D: 3
D: 2
D: 1
D: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

Priority

```
daniel@daniel-VirtualBox://home/daniel/downloads/nachos-4.0/code/threads$ ./nachos PRIORITY
Priority
D: 8
D: 7
D: 6
D: 5
D: 4
D: 3
D: 2
D: 1
D: 0
C: 3
C: 2
C: 1
C: 0
A: 2
A: 1
A: 0
B: 9
B: 8
B: 7
B: 6
B: 5
B: 4
B: 3
B: 2
B: 1
B: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

SJF

```
daniel@daniel-VirtualBox://home/daniel/downloads/nachos-4.0/code/threads$ ./nachos SJF
SJF
A: 2
A: 1
A: 0
C: 3
C: 2
C: 1
C: 0
D: 8
D: 7
D: 6
D: 5
D: 4
D: 3
D: 2
D: 1
D: 0
B: 9
B: 8
B: 7
B: 6
B: 5
B: 4
B: 3
B: 2
B: 1
B: 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

Test with test1 and test2 (FCFS)

```
daniel@daniel-VirtualBox://home/daniel/downloads/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2 FCFS
FIFO
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:6
return value:0
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

3. Result

When calling Sleep(), the thread indeed goes to sleep, but don't sleep for correct amount of time. For example, in sleep.c, the thread is put to sleep for 1000ms. But the thread only slept for less than one second. I guess I didn't fully understand how Alarm works.

In CPU scheduling, I can't successfully run FCFS method. When using FCFS, produced result will be same as using RR. Then I realize when using FCFS, I can't disable the time slicing performed by Alarm. So, I use brutal force to solve this problem. I disable alarm when using FSFC to initialize a thread.