作業系統 Report

作業一 R10945061 林宇恆

1.Motivation:

根據作業要求敘述，定義 Sleep()的 system call，下圖為根據敘述在 start.s 中以 instruction 定義暫存器存取 Sleep()函式，直接依樣畫葫蘆先前的函式，要注意.globl 和.ent 要上下連接，否則會有 error。

```
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
```

而在 system.h 中定義變數 SC_Sleep 跟函式 void Sleep()。

```
#define SC_ThreadFork   9
#define SC_ThreadYield  10
#define SC_PrintInt 11
#define SC_Sleep        12
#ifndef IN_ASM

void PrintInt(int number);  //my System Call
void Sleep(int number);  //define the sleep()
```

依照作業要求，將 thread 進入休眠，設計使用 sleep list 依照起床時間小於預期就將其從 list 中剔除，當 Sleep()時，會呼叫 WaitUntil()並丟入 list 睡覺，在 CallBack()呼叫後去 list 中檢查誰要起床。

而排程的部分再多個演算法選擇中在 main.cc 中設計選擇的方法，我是以在 terminal 增加引數的方式去選擇當下使用的演算法。

```
SchedulerType type = RR;
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;
}
```

根據助教說明在 kernel.cc 中添加 self test 添加自己測試的函式，我定義在 thread.cc 中，用來測試排程是否正確

```
currentThread->SelfTest();   // test thread switching
Thread::SchedulingTest();
```

```
void
Thread::SchedulingTest()
{
    const int thread_num = 4;
    char *name[thread_num] = {"A", "B", "C", "D"};
    int thread_priority[thread_num] = {5, 1, 3, 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();
}
```

2. Implementation:

Sleep()實作部分在 exception.cc 中添加 Sleep 的例外操作，並根據助教提示利用 kernel->alarm->WaitUntil()

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

並且在 alarm.cc 中只有實作了 CallBack()，需要建立一個 class 來存取睡眠中的 thread 和各自的起床時間，被 interrupt 後哪些 thread 需要起來，按照時間順序使用了內建的 sortlist。函數判斷 empty 以及存取時間跟緒等等，controler

則是決定目前哪個 thread 該起床且 timing 為何時。

```
class controler;
class sleep_mode{
    public:
    sleep_mode(){
        interupt_num = 0;
}
    void forcesleep(Thread* t, int x);
    bool wakeup();
    bool isempty();
    private:


    int interupt_num;
    list<controler> nums;
};
class controler{
            public:
                controler(Thread *t, int x){
                sleeper = t;
                timing = x;
        }
            Thread* sleeper;
            int timing;
        };
```

```
bool sleep_mode::isempty(){
    return (nums.size());
}
void sleep_mode::forcesleep(Thread* t,int x){
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    nums.push_back(controler(t,interupt_num + x));
    t->Sleep(false);
}
bool sleep_mode::wakeup(){
    bool token1 = false;
    interupt_num++;
    for(list<controler>::iterator it=nums.begin();it!=nums.end();){
        if(interupt_num >= it->timing){
            token1 = true;
            kernel->scheduler->ReadyToRun(it->sleeper);
            it = nums.erase(it);
        }
        else{
            it++;}
    }
return token1;
}
void Alarm::WaitUntil(int x){
    IntStatus old = 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;
    sleeping.forcesleep(t,x);
    kernel->interrupt->SetLevel(old);
    return;
}
```

CPU Scheduling 中測試部分需要有兩種 case，因此設計引數來針對不同演算法進行不同的 test，在 scheduler.h 中維持原來設定 RR,SJF 以及 priority，唯一不同是設計以數，因此將 Scheduler 的建構子設定為輸入參數，讓建構子能夠決定使用哪種排程。

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

class Scheduler {
  public:
    Scheduler();        // Initialize list of ready threads
    ~Scheduler();               // De-allocate ready list
        Scheduler(SchedulerType type);
```

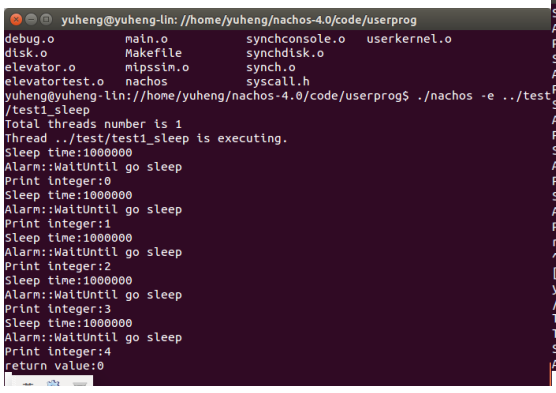
設計演算法如下：

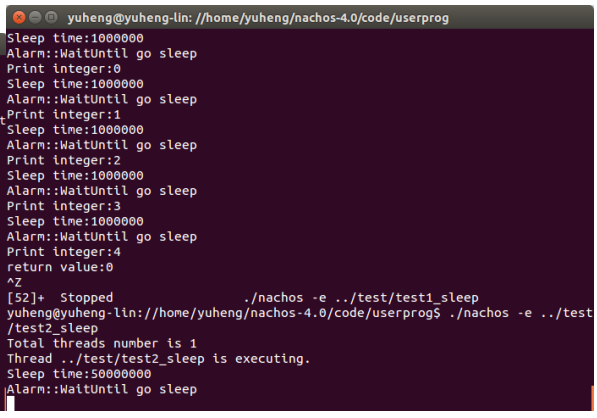
```
int SJFCompare(Thread *a, Thread *b) {
    if(a->getBurstTime() == b->getBurstTime())
        return 0;
    return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
}
int PriorityCompare(Thread *a, Thread *b) {
    if(a->getPriority() == b->getPriority())
        return 0;
    return a->getPriority() > b->getPriority() ? 1 : -1;
}
Scheduler::Scheduler()
{
    schedulerType = RR;
    readyList = new List<Thread *>;
    toBeDestroyed = NULL;

}
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;
  }toBeDestroyed = NULL;
}
```

3. Result:

Sleep()執行結果如下，要注意 makefile 中也要進行修改在 make 後才會跑出 test 的.o 檔






```
#include "syscall.h"
main()
    {
        int n;
        for (n = 0;n < 5; n++){
            Sleep(1000000);
            PrintInt(n);
}
    }
```

```
#include "syscall.h"
main()
    {
        int n;
        for (n=0;n<10;n--){
            Sleep(50000000);
            PrintInt(50);
}
    }
```

Makefile 修改如下：

```
test1_sleep: test1_sleep.o start.o
	$(LD) $(LDFLAGS) start.o test1_sleep.o -o test1_sleep.coff
	../bin/coff2noff test1_sleep.coff test1_sleep
test2_sleep: test2_sleep.o start.o
	$(LD) $(LDFLAGS) start.o test2_sleep.o -o test2_sleep.coff
	../bin/coff2noff test2_sleep.coff test2_sleep
```

CPU Scheduling 執行結果如下

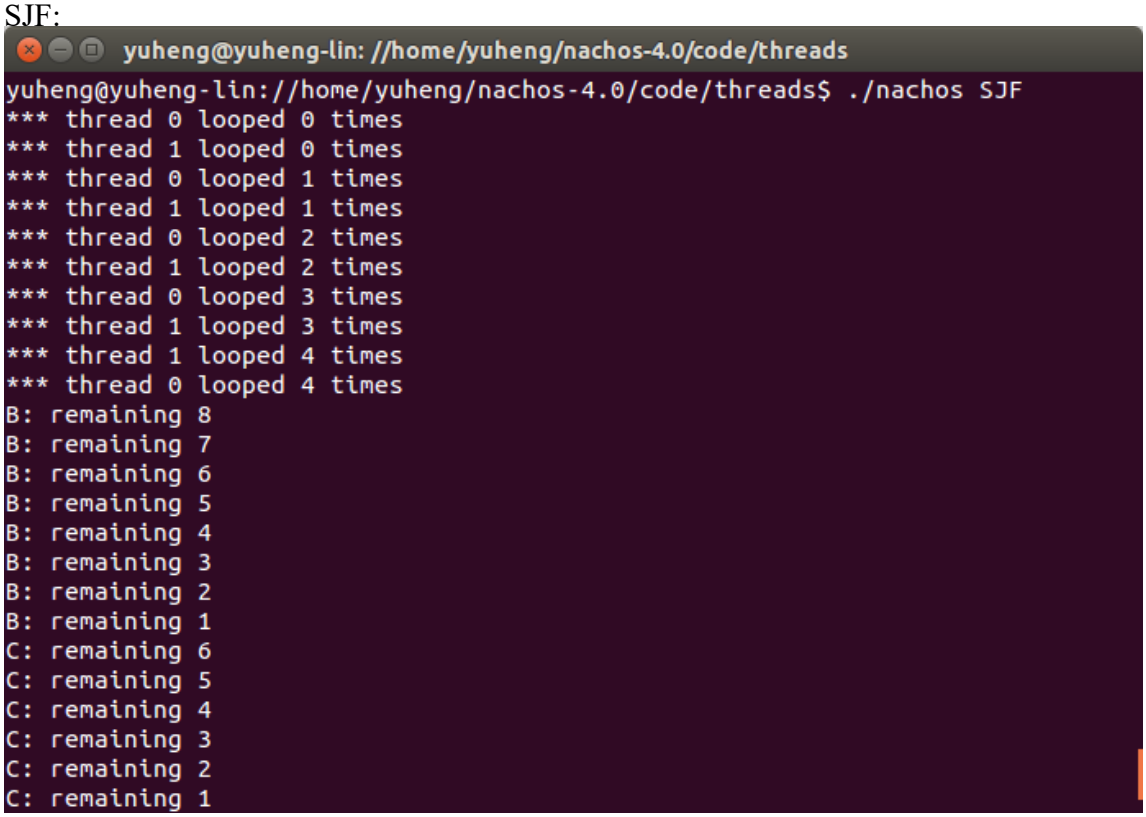
RR:

```
yuheng@yuheng-lin: /home/yuheng/nachos-4.0/code/threads
yuheng@yuheng-lin:/home/yuheng/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
```

Priority:

```
yuheng@yuheng-lin: /home/yuheng/nachos-4.0/code/threads
yuheng@yuheng-lin:/home/yuheng/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
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
```

SJF:


yuheng@yuheng-lin: //home/yuheng/nachos-4.0/code/threads
yuheng@yuheng-lin://home/yuheng/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 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