HW 1: Implement SYSCALL sleep

分工:

B10615045 陳尚富: 主要撰寫 HW1

B10601019 鞠傳豐: 討論隨機分組前，他們未完成的部分

B10601033 陳彥憑: 討論隨機分組前，他們未完成的部分

這次作業系統分組來的太突然，我認識的朋友們都在我睡著的時候偷偷分完組了，所以我去隨機分組。然後隨機分組的時機又有點晚，所以在確認隊員前，就已經把作業連夜趕出來了。導致分組後，沒有完善的分工，所以我和新隊員們，就討論我寫的 code 和他們分組前未完成的部分。

想法:

本次作業主要更改的檔案有:

Nachos/code/threads/scheduler.cc (.h)

Nachos/code/threads/thread.cc (.h)

Nachos/code/threads/alarm.cc (.h)

在經過一段時間的看 code 後，我認為 Nachos 沒有把 Block 狀態完成。所以我先在 Schedule 新增了一個存放 Block Thread 的 list，用來存放被 sleep 的 thread。還有一些 Block 時或是從 Block list 喚醒時會需要的 function。

```
  /* Writen bu @shungfu */
  void Blocked(Thread * thread);  // Thread are being Blocked
  bool WakeUp();  // Check if thread can be ReadyToRun
  void PopBlock(Thread*thread){blockList->remove(thread);}
  bool IsBlockEmpty(){return blockList->empty();}
private:
  SchedulerType schedulerType;
  List<Thread *> *readyList;  // queue of threads that are ready to run,
                              // but not running
  Thread *toBeDestroyed;      // finishing thread to be destroyed
                              // by the next thread that runs

  /* Writen by @shungfu */
  std::list<Thread *> *blockList; // queue of threads that are blocked
```

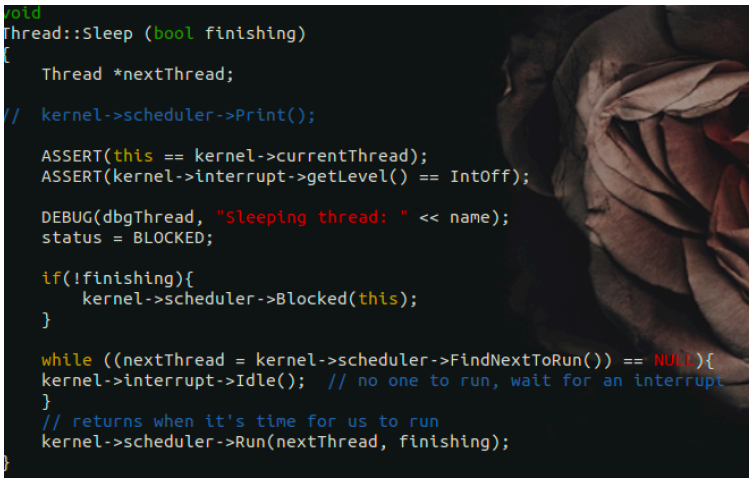
接著我認為，那些 sleep 的 thread 和 thread 在本質上應該都還是一樣的東西，所以我直接在 Thread 的 class 裡面增加了一個 private member sleep_time，用來記錄這個 thread 會睡多久，還有一些與其相關的 function。

```
  /* Written by @shungfu */
  void SetSleepTime(int time);
  int getSleepTime(){return sleep_time;}
private:
  // some of the private data for this class is listed above

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

  void StackAllocate(VoidFunctionPtr func, void *arg);
                      // Allocate a stack for thread.
                      // Used internally by Fork()

  /* Writen by @shungfu */
  int sleep_time;
```

在完成上述後，我就開始實作並規劃 class Alarm 裡的 WaitUntil。在呼叫 syscall sleep 之後，我會執行 Alarm 的 WaitUntil，然後再 function 內先設定 sleep 的時間(sleep 時間為當前的 totalTicks + 傳入的 time)，

```
//----------------------------------
// Alarm::WaitUntil()
// Suspend execution until time > now + x
// **Writen by @shungfu**
//----------------------------------

void
Alarm::WaitUntil(int x)
{
    // turn off the interrupt
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);

    Thread *thread = kernel->currentThread;
    thread->SetSleepTime(x);

    // Make the thread Sleep
//  cout << "Thread goes Sleep" << endl;
    thread->Sleep(false);

    // turn on the interrupt
    kernel->interrupt->SetLevel(oldLevel);
}
```

緊接著 call Thread 的 Sleep。在 class Thread 裡面才會把 thread 狀態設定為 Blocked 然後將其推入我創建的 BlockList。



```
void
Thread::Sleep (bool finishing)
{
    Thread *nextThread;

//  kernel->scheduler->Print();

    ASSERT(this == kernel->currentThread);
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    DEBUG(dbgThread, "Sleeping thread: " << name);
    status = BLOCKED;

    if(!finishing){
        kernel->scheduler->Blocked(this);
    }

    while ((nextThread = kernel->scheduler->FindNextToRun()) == NULL){
    kernel->interrupt->Idle();  // no one to run, wait for an interrupt
    }
    // returns when it's time for us to run
    kernel->scheduler->Run(nextThread, finishing);
}
```

```
//------------------------------
// Writen by @shungfu
// Threads that being Blocked
//------------------------------
void
Scheduler::Blocked(Thread * thread)
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    DEBUG(dbgThread,"Putting thread on Block List: " << thread->getName());

    thread->setStatus(BLOCKED);
    blockList->push_back(thread);
}
```

然後在每次系統呼叫到 Alarm::CallBack 時，去檢查有沒有 Block 的 thread 需要被喚醒。有的話就將其重新推回 readyList 裡面，被等待執行。

```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();

    Thread *thread;
    bool WakeUp = false;

    // if someone to wakeup, then put it back to ready list and pop
    if(kernel->scheduler->WakeUp()){
        WakeUp = true;
    }

    // Check also Block status, and if someone wakeup do context switch
    if (status == IdleMode && kernel->scheduler->IsBlockEmpty() && !WakeUp) {    // is it time to quit?
        if (!interrupt->AnyFutureInterrupts()) {
        timer->Disable();   // turn off the timer
    }
    } else {            // there's someone to preempt
    interrupt->YieldOnReturn();
//  cout << "Total Ticks: " << kernel->stats->totalTicks << endl;
    }
}
```

```
//----------------------
// Check if thread can be ReadyToRun
//----------------------
bool
Scheduler::WakeUp()
{
    bool toReady = false;
    Thread* now;
    std::list<Thread*>::iterator ptr = blockList->begin();
    // check every thread, if there's someone to wakeup
    for(; ptr != blockList->end();)
    {
        // Time to WakeUp, put it to ready list and pop it
        if((*ptr)->getSleepTime() <= kernel->stats->totalTicks){
            now = *ptr;
            ptr++;
            kernel->scheduler->ReadyToRun(now);
            kernel->scheduler->PopBlock(now);
            toReady = true;
            continue;
        }
        ptr++;
    }
    return toReady;
}
```

執行結果：

```
shungfu@ubuntu:~/NachOS/code/userprog$ ./nachos -e ../test/sleep
Total threads number is 1
Thread ../test/sleep is executing.
Sleep Value:380000000
Total Ticks: 49, sleep until: 380000049
Print integer:222
Sleep Value:380000000
Total Ticks: 380000143, sleep until: 760000143
Print integer:222
Sleep Value:380000000
Total Ticks: 760000243, sleep until: 1140000243
Print integer:222
Sleep Value:380000000
Total Ticks: 1140000343, sleep until: 1520000343
Print integer:222
Sleep Value:380000000
Total Ticks: 1520000443, sleep until: 1900000443
Print integer:222
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1900000600, idle 1900000330, system 130, user 140
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
shungfu@ubuntu:~/NachOS/code/userprog$
```

遭遇難題：

在測試 syscall Sleep 時發現，執行時 nachos 會去檢查 ready list 是不是為空，如果是的話，他就會加速來檢查會不會有新的 interrupt 發生。而且我也不太確定 nachos 的一個 tick 到底是幾秒。所以在時間設 500000 以下的情況時，nachos 快到讓我感覺像是一瞬間發生，所以我就將時間條大到 50000000。但在這個情況下，倘若我執行五次迴圈，nachos 的 totalTicks 就會發生 overflow，導致程式提前結束。