OS_Final_Project

Part 1. Trace Code

1-1. New → Ready

```
UserProgKernel::InitializeAllThreads();
/* Explain: 1. 為每個將要執行的程式配上一個 thread。
*/

UserProgKernel::InitializeOneThread(char*, int, int);
/* Explain: 1. 創建一個 thread 並初始化
                        2. 透過 Fork 賦予 thread 任務
*/

Thread::Fork(VoidFunctionPtr, void*);
/* Explain: 1. 透過 StackAllocate 為 thread 分配 stack空間
                        2. Disable interrupt
                        3. 透過 ReadyToRun 將 thread 放進 ReadyQu
                        4. 將 interrupt 改回原本的狀態
*/

Thread::StackAllocate(VoidFunctionPtr, void*);
/* Explain: 1. 為 thread 分配並初始化 stack 空間,
               並針對不同處理器架構有不同的初始化方式。
*/

Scheduler::ReadyToRun(Thread*);
/* Explain: 1. 根據 priority 將其 thread 放進對應的 ReadyQueue
                        2. 更新 RemainingBurstTime, status, Wait
                        3. 如果是加進 L1 ReadyQueue 則需要檢查有沒有
                           如果有搶奪，則 call Thread:Yield() 處理
*/
```

1-2. Running → Ready

```
Machine::Run();
/* Explain: 1. 切換成 user mode 解碼並執行一條指令
                        2. 透過 OneTick() 模擬時間
*/

Interrupt::OneTick();
/* Explain: 1. 模擬時間
                        2. Disable interrupt
                        3. 檢查有沒有 pending interrupts ，如果有 p
                           ，則順便處理 pending interrupts
                        4. Enable interrupt
                        5. 如果有需要做 context switch ，則切換成 s
                        6. 改回原本的狀態 (user or system mode)
*/

Thread::Yield();
/* Explain: 1. Disable interrupt
                        2. 終止現在正在跑的 thread 並根據其 priority
                        3. 透過 FindNextToRun 找下一個要執行的 thre
                        4. Reset current_thread 的 RemainingBurs
                        5. 透過 Run 執行 thread
                        6. 將 interrupt 改回原本的狀態
*/

Scheduler::FindNextToRun();
/* Explain: 1. 依序檢查 L1ReadyQueue, L2ReadyQueue, L3ReadyQueue
                           如果有任一 ReadyQueue 非空，則 Remove 最
                           並且 return 該 thread
*/

Scheduler::ReadyToRun(Thread*);
/* Explain: 1. 根據 priority 將其 thread 放進對應的 ReadyQueue
                        2. 更新 thread 的 status, WaitTime 等等
```

```
                        3. 如果是加進 L1 ReadyQueue 則需要檢查有沒有
                           如果有搶奪，則 call Thread:Yield() 處理
*/

Scheduler::Run(Thread*, bool);
/* Explain: 1. bool 用來表示 oldthread 是否執行完畢。如果執行完畢則標記
                        2. oldthread 和 nextthread 做 context sw
                        3. 如果 oldthread 有被標記，則刪除 oldthrea
                        4. 執行 nextthread
*/
```

1-3. Running → Waiting

```
ExceptionHandler(ExceptionType) case SC_PrintInt
/* Explain: 1. 輸出 Register4 裡面的整數
*/

SynchConsoleOutput::PutInt();
/* Explain: 1. 將整數轉換為 str
                        2. lock->Acquire(); 對輸出做互斥存取
                        3. 重複輸出 char 直到 str 輸出完畢
                           其中 waitFor->P() 等待單個 char 輸出完畢
                        4. lock->Release();
*/

ConsoleOutput::PutChar(char);
/* Explain: 1. putBusy 用來記錄有沒有其他 char 正在被輸出
                        2. 透過 interrupt::Schedule 將上述任務加進
                           未來等待 OneTick() 被執行時一併處理。
*/

Semaphore::P();
/* Explain: 1. Disable interrupt
                        2. 持續檢查 value 值是否為0
                           如果 value 值為 0，則把該 thread 丟進 qu
```

```
                              並且透過 Thread::Sleep(False) 處理後續的
                              False 代表該 thread 尚未執行完畢
                           3. Semaphore 的值 - 1
                           4. Enable interrupt
*/

SynchList<T>::Append(T);
/* Explain: SynchList 用來管理有哪些 thread 是 blocked 的狀態
                           1. lock->Acquire(); 對 list 做互斥存取
                           2. 將 T 加進 list 中
                           3. 如果有等待該 list 非空的 waiters 則 wake
                           4. lock->Release();
*/

Thread::Sleep(bool);
/* Explain: 1. bool 用來表示 oldthread 是否執行完畢
                           2. 透過 FindNextToRun 找下一個要執行的 next
                              直到發生 interrupt
                           3. 更新 RemainingBurstTime, status, RunT
                           4. 透過 Run 執行 nextthread
*/

Scheduler::FindNextToRun();
/* Explain: 1. 依序檢查 L1ReadyQueue, L2ReadyQueue, L3ReadyQueue
                              如果有任一 ReadyQueue 非空，則 Remove 最
                              並且 return 該 thread
*/

Scheduler::Run(Thread*, bool);
/* Explain: 1. bool 用來表示 oldthread 是否執行完畢。如果執行完畢則標記
                           2. oldthread 和 nextthread 做 context sw
                           3. 如果 oldthread 有被標記，則刪除 oldthrea
                           4. 執行 nextthread
*/
```

1-4. Waiting → Ready

```
Semaphore::V();
/* Explain: 1. Disable interrupt
                      2. 檢查 queue 裡面有沒有先前被 block 的 threa
               如果有的話，則將位於 queue 最前面的 thread 加進 ReadyQ
                        3. Semaphore的值 + 1
                        4. Enable interrupt
*/

Scheduler::ReadyToRun(Thread*);
/* Explain: 1. 根據 priority 將其 thread 放進對應的 ReadyQueue
                        2. 更新 thread 的 status, WaitTime 等等
                        3. 如果是加進 L1 ReadyQueue 則需要檢查有沒有
                           如果有搶奪，則 call Thread:Yield() 處理
*/
```

1-5. Running → Terminated

```
ExceptionHandler(ExceptionType) case SC_Exit
/* Explain: 1. 告訴 OS thread 執行完畢
*/

Thread::Finish();
/* Explain: 1. Disable interrupt
                        2. Call Thread::Sleep(True)
               True 表示該 thread 已執行完畢
*/

Thread::Sleep(bool);
/* Explain: 1. bool 用來表示 oldthread 是否執行完畢
                        2. 透過 FindNextToRun 找下一個要執行的 next
                           直到發生 interrupt
                        3. 更新 RemainingBurstTime, status, Runt
```

```
                          4. 透過 Run 執行 nextthread
*/

Scheduler::FindNextToRun();
/* Explain: 1. 依序檢查 L1ReadyQueue, L2ReadyQueue, L3ReadyQueue
                             如果有任一 ReadyQueue 非空，則 Remove 最
                             並且 return 該 thread
*/

Scheduler::Run(Thread*, bool);
/* Explain: 1. bool 用來表示 oldthread 是否執行完畢。如果執行完畢則標記
                          2. oldthread 和 nextthread 做 context sw
                          3. 如果 oldthread 有被標記，則刪除 oldthrea
                          4. 執行 nextthread
*/
```

1-6. Ready → Running

```
Scheduler::FindNextToRun();
/* Explain: 1. 依序檢查 L1ReadyQueue, L2ReadyQueue, L3ReadyQueue
                             如果有任一 ReadyQueue 非空，則 Remove 最
                             並且 return 該 thread
*/

Scheduler::Run(Thread*, bool);
/* Explain: 1. bool 用來表示 oldthread 是否執行完畢。如果執行完畢則標記
                          2. oldthread 和 nextthread 做 context sw
                          3. 如果 oldthread 有被標記，則刪除 oldthrea
                          4. 執行 nextthread
*/

SWITCH(Thread*, Thread*);
/* Explain: 1. 先把 registers (%eax, %ebx, %ecx, %edx, %esi, %ed
                               存到 thread t1 的 stack (分別存到 _EA
                               _EDX(esp_t1), _ESI(esp_t1), _EDI(es
```

```
                                wherer esp_t1 代表 t1 的 stack point

                                做法: 用 _eax_save 先存 %eax ，後續用
                                然後把其他的 registers 的值分別存到對應

                          2. 把 return address 存到 _PC(esp_t1)
                          3. 把 stack 裡面儲存 thread t2 的資料 (_EA
                                _EDX(espt_2), _ESI(esp_t2), _EDI(es
                                load 到 registers (分別 load 到 %eax
                                where esp_t2 代表 t2 的 stack pointe

                                做法: 用 %eax 存取 esp_t2 (在 code 中
                                然後再把 stack 的資料 load 到對應的 re
                          4. 最後在把 return address 存到 stack 的頂
                                (看圖的話是存到 12(%esp)
                                看 code 的話，因為前面 movl    _ESP(%
                                %esp 的值已經 +8 了，所以後續只要在補 4
                                圖↓
                                8(esp)  ->              thread *t2
                4(esp)  ->              thread *t1
                (esp)  ->              return address
*/

for loop in Machine::Run();
/* Explain: 重複執行
                          1. 切換成 user mode 解碼並執行一條指令
                          2. 透過 OneTick() 模擬時間
*/
```

Part 2. Code Implementation

在 `Scheduler` 類別的構造函數中，我們初始化了三個準備隊列

```
Scheduler::Scheduler()
{
```

```
    // L1 uses preemptive SRTN, sort by remaining burst time
    L1ReadyQueue = new SortedList<Thread*>(CompareRemainingBurst

    // L2 uses FCFS, sort by thread ID
    L2ReadyQueue = new SortedList<Thread*>(CompareThreadID);

    // L3 uses round-robin
    L3ReadyQueue = new List<Thread*>();

    toBeDestroyed = NULL;
}
```

根據線程的優先級，我們將線程插入不同的準備隊列。在 `ReadyToRun` 函數中實現了這個過程：

```
void Scheduler::ReadyToRun(Thread *thread)
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    int queueLevel = -1;
    int priority = thread->getPriority();

    if (priority >= 100 && priority <= 149) {
        L1ReadyQueue->Insert(thread);
        queueLevel = 1;
        // Check for preemption
        Thread* currentThread = kernel->currentThread;
        if (currentThread != NULL && currentThread->getPriority(
            if (thread->getRemainingBurstTime() < currentThread-
                kernel->currentThread->Yield();
            }
        }
    } else if (priority >= 50 && priority <= 99) {
        L2ReadyQueue->Insert(thread);
        queueLevel = 2;
    } else if (priority >= 0 && priority <= 49) {
```

```
        L3ReadyQueue->Append(thread);
        queueLevel = 3;
    }

    DEBUG('z', "[InsertToQueue] Tick [" << kernel->stats->total

    thread->setStatus(READY);
}
```

在 `FindNextToRun` 函數中，我們從準備隊列中選擇下一個要運行的線程：

```
Thread * Scheduler::FindNextToRun ()
{
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    Thread* nextThread = NULL;
    int queueLevel = -1;

    if (!L1ReadyQueue->IsEmpty()) {
        nextThread = L1ReadyQueue->RemoveFront();
        queueLevel = 1;
    } else if (!L2ReadyQueue->IsEmpty()) {
        nextThread = L2ReadyQueue->RemoveFront();
        queueLevel = 2;
    } else if (!L3ReadyQueue->IsEmpty()) {
        nextThread = L3ReadyQueue->RemoveFront();
        queueLevel = 3;
    }

    if (nextThread) {
        DEBUG('z', "[RemoveFromQueue] Tick [" << kernel->stats->
    }
    return nextThread;
}
```

我們在 `UpdatePriority` 函數中定期更新線程的優先級和等待時間，以避免饑餓現象並實現年老化策略：

```
void Scheduler::UpdatePriority()
{
    updateQueuePriority(L1ReadyQueue, 1);
    updateQueuePriority(L2ReadyQueue, 2);
    updateQueuePriority(L3ReadyQueue, 3);
}

void Scheduler::updateQueuePriority(List<Thread*>* queue, int qu
    ListIterator<Thread*> *iter = new ListIterator<Thread*>(queu
    Thread* thread;
    int oldPriority, newPriority;

    for (; !iter->IsDone(); iter->Next()) {
        thread = iter->Item();
        if (kernel->stats->totalTicks - thread->getWaitTime() >
            oldPriority = thread->getPriority();
            newPriority = oldPriority + 10;
            DEBUG('z', "[UpdatePriority] Tick [" << kernel->stat

            if (newPriority > 149) newPriority = 149;

            thread->setPriority(newPriority);
            thread->setWaitTime(kernel->stats->totalTicks);
            queue->Remove(thread);
            ReadyToRun(thread);
        }
    }
    delete iter;
}
```