實驗六 Thread Synchronization by Barrier API

9617145 資工 4C 許晏峻

1. 實驗目的

學習使用 pthread_mutex_lock、pthread_cond_wait 和 pthread_cond_signal 的概念。

2. 實驗環境

測試數據環境如下：

- 實體機器
- CPU：AMD Phenom(tm) II X6 1055T Processor
- Core：6
- Memory：8G
- OS：Ubuntu 10.04.02 TLS x86_64

3. 步驟過程

3.1. polling

檢查 flag 設置位置，可以發現從邏輯上，producer 的 fifo->turn 賦值應在 for 迴圈外，而不是內，因此將 fifo->turn = CONSUMER_TURN; 和 fifo->is_empty = false; 放置迴圈外即可。

時間計算結果錯誤：printf 中，diff_time:%lld.%lld\n 應該成 diff_time:%lld.%09lld\n，否則若小數點小於 0.1，輸出會錯誤；此外 printf 中%09lld 對應的變數應該為 end_time.tv_nsec - start_time.tv_nsec 而不是 end_time.tv_sec - start_time.tv_nsec。

compile 時的警告：pthread_create 時的第三、四個參數可先自行做型態轉換成 (void *)；printf 時使用的 %lld 可將對應的變數自行轉換成 (long long int)。

3.2. lock

使用 pthread_mutex_t 型態對特定的變數做 lock 保護，並且設定兩個 pthread_cond_t 型態來對 consumer 和 producer 作流程的控制。

首先先在 producer 迴圈外做 lock，接著以一個 global variable：counter 當 counter 計算目前有幾條 thread 的 producer 已經做好，接著若所有 producer 都完成，即呼叫 signal 啟動 consumer，之後呼叫 wait 等待這一輪的 consumer 完成。

而在 consumer 部分，首先進入 function 時先使用 lock 並且呼叫 wait 等待所有 producer 都做完並且呼叫 signal；當 producer 呼叫 signal 啟動 consumer 時，就開始工作直到 for 迴圈做完，接著將 counter 歸零以便下一輪 producer 計算 thread 完成數，最後呼叫 broadcast 來啟動所有 producer 進行下一輪的迴圈。

3.3. barrier

自行時做一個 barrier structure，其中包含一個 pthread_mutex_t 型態的 lock 來保護特定變數，並設定兩個 pthread_cond_t 型態來對 consumer 和 producer 作流程的控制，最後在設定一

個 int 值來計算 thread 完成 producer 的數量。

接者實作一個 function 來對 barrier structure 做初始化。

最後實作一個 function 來對所有 producer 做 barrier 流程控制，如下敘述：

進入 function 後先上 lock，如果這個 barrier 代表剛做完 consumer 的工作，就會將 counter 歸零，並且呼叫 broadcast 來啟動所有 producer 來進行下一輪迴圈，解除 lock；而如果這個 barrier 代表剛做完其中一個 producer，將 counter 加一，並判斷是否所有 producer 都做完，若是，則呼叫 signal 啟動 consumer，之後呼叫 wait 等待這一輪的 consumer 完成。

在 producer 部分，只要在 for 迴圈做完後加上一個 barrier function 即可。

而在 consumer 部分，進入後必須先上 lock 並且呼叫 wait 等待 barrier 呼叫 signal，而在 for 迴圈外也只須加上一個 barrier function 來啟動所有 producer。

4. 數據結果

	polling	lock	barrier
./main 2 1000	0.0480147 secs	0.153203093 secs	0.15425128 secs
./main 2 10000	0.315558696 secs	1.551633949 secs	1.554473001 secs

```
ychsu-workstation [/home/ychsu/Work/MP/lab6/polling] -ychsu- % make run
./main 2 1000
NUM_THREADS:2, NUM_LOOPS:1000
sig=152ff258
Main: program completed. Exiting.
s_time.tv_sec:1302668898, s_time.tv_nsec:829353124
e_time.tv_sec:1302668898, e_time.tv_nsec:877367824
diff_time:0.048014700
```

圖 1 polling (1000)

```
ychsu-workstation [/home/ychsu/Work/MP/lab6/polling] -ychsu- % make run
./main 2 10000
NUM_THREADS:2, NUM_LOOPS:10000
sig=621fded4
Main: program completed. Exiting.
s_time.tv_sec:1302668967, s_time.tv_nsec:829425944
e_time.tv_sec:1302668968, e_time.tv_nsec:144984640
diff_time:0.315558696
```

圖 2 polling (10000)

```
ychsu-workstation [/home/ychsu/Work/MP/lab6/lock] -ychsu- % make run
./main 2 1000
NUM_THREADS:2, NUM_LOOPS:1000
sig=152ff258
Main: program completed. Exiting.
s_time.tv_sec:1302669028, s_time.tv_nsec:159370353
e_time.tv_sec:1302669028, e_time.tv_nsec:312573446
diff_time:0.153203093
```

圖 3 lock (1000)

```
ychsu-workstation [/home/ychsu/Work/MP/lab6/lock] -ychsu- % make run
./main 2 10000
NUM_THREADS:2, NUM_LOOPS:10000
sig=621fded4
Main: program completed. Exiting.
s_time.tv_sec:1302669099, s_time.tv_nsec:009419390
e_time.tv_sec:1302669100, e_time.tv_nsec:561053339
diff_time:1.551633949
```

圖 4 lock (10000)

```
ychsu-workstation [/home/ychsu/Work/MP/lab6/barrier] -ychsu- % make run
./main 2 1000
NUM_THREADS:2, NUM_LOOPS:1000
sig=152ff258
Main: program completed. Exiting.
s_time.tv_sec:1302669205, s_time.tv_nsec:419452902
e_time.tv_sec:1302669205, e_time.tv_nsec:573704182
diff_time:0.154251280
```

圖 5 barrier (1000)

```
ychsu-workstation [/home/ychsu/Work/MP/lab6/barrier] -ychsu- % make run
./main 2 10000
NUM_THREADS:2, NUM_LOOPS:10000
sig=621fded4
Main: program completed. Exiting.
s_time.tv_sec:1302669159, s_time.tv_nsec:589423692
e_time.tv_sec:1302669161, e_time.tv_nsec:143896693
diff_time:1.554473001
```

圖 6 barrier (10000)

5. 結論心得

由數據結果，polling 和 lock 版本跟在實驗課上所執行的速度相反過來，個人猜測也許和 cpu 有關係，使得 busy loop 沒有造成過大的 loading，反而是不斷的 lock、unlock、wait、signal 造成的 loading 較大。而 barrier 版本和 lock 版本速度差異不大是因為實作構想是差不多的，只是 barrier 有包裝成自訂的 structure 和使用自訂的 function 來控制 singal 和 wait 而已。

這次實驗，最重要是學到了 signal 和 wait 的概念，如何使用 lock 搭配 wait 作流程控制，而 lock 和 unlock 一定要成對，並且要注意 wait 和 signal 是否有先後順序的問題：signal 先到不會保留，要有 wait 去對應，否則 wait 晚到會造成 starvation。