Assignment 3 Answer & Report

生機四 B10605023 詹育晟

6.14

(a) (Race Conditions)

1. 共享變數

本題的唯一共享資源為全域變數 number_of_processes，它在 allocate_process() 與 release_process() 兩個函式中都會被讀寫。

2. 主要競態場景

- **分配階段的檢查與更新不具原子性**
  - 先判斷 number_of_processes == MAX_PROCESSES，再執行 ++number_of_processes。
  - 若兩個執行緒同時執行到判斷句，都因尚未達上限而通過，接著再同時做自增，就會造成：
    1. **超出上限**（例如從 254→256），導致資源耗盡卻仍允許分配。
    2. **計數錯漏**（例如兩次自增只累加了一次），造成分配過程中 PID 丟失。
- **分配與釋放交錯更新**
  - 一條執行緒正在 --number_of_processes，另一條同時做 ++number_of_processes，就可能讀到錯誤的中間值。
  - 例如：初始為 1，執行緒 A 減到 0，執行緒 B 再加到 1，最終回到 1，但實際上應該是「釋放一個→再分配一個」，結果仍顯示 1，錯誤地掩蓋了中間的變化。

3. 為何構成競態

- 對同一個全域變數的「讀取」與「寫入」沒有互斥保護，導致多執行緒同時進入檢查與更新程式區段時，相互干擾、產生不一致結果。
- 這種「先讀後寫」被其他執行緒插入的交錯行為，就是典型的競態條件。

(b) 加鎖位置說明

為了避免多執行緒並發存取及修改全域變數 number_of_processes 而產生競態，在所有「讀取／更新」該變數的程式區段前後都必須使用同一把互斥鎖 mutex：

1. **在 allocate_process() 中**
    - **進入臨界區**：最一開始進入函式、進行任何檢查前，呼叫 acquire(&mutex)。
    - **檢查與遞增**：在鎖住狀態下先做 if (number_of_processes == MAX_PROCESSES) 檢查，再進行 ++number_of_processes 更新。
    - **離開臨界區**：完成遞增後，呼叫 release(&mutex)，讓其他執行緒能夠進入。
2. **在 release_process() 中**
    - **進入臨界區**：呼叫 acquire(&mutex)，確保此時只有該執行緒能存取 number_of_processes。
    - **遞減更新**：在鎖住狀態下執行 --number_of_processes。
    - **離開臨界區**：更新完成後，呼叫 release(&mutex)。

6.33

1. 作業處理邏輯

1. **目標**：使用 Monte Carlo 方法估算 π 值。
2. **多執行緒設計**：
    - 建立 5 個執行緒（threadsNum = 5），每個執行緒各自產生 1,000 個隨機點（pointsPerThread = 1000）。
    - 每個點都隨機落在 [-1, +1]×[-1, +1] 的正方形範圍內。
    - 判斷該點是否落在單位圓內（$x^2 + y^2 \leq 1$），如果是，累加全域變數 counter。
3. **互斥保護**：
    - 所有執行緒共用同一個全域計數器 counter。
    - 在每次「發現落在圓內」要修改 counter 時，都要先 pthread_mutex_lock(&mutex)，完成遞增後再 pthread_mutex_unlock(&mutex)。
    - 這樣可以避免多執行緒同時修改 counter 時發生競態條件。
4. **結果彙整**：

- 主執行緒等待所有 worker thread 完成（pthread_join）。
- 最後，透過公式計算出 π 的近似值並輸出。

2. 程式執行流程

程式一開始宣告並初始化全域計數器 counter = 0，同時以 PTHREAD_MUTEX_INITIALIZER 建立一把互斥鎖。在 main() 中設定 5 個執行緒、每個執行緒產生 1000 個隨機點（動態配置 tids 存放各執行緒 ID，並呼叫 pthread_attr_init() 初始化屬性），接著以迴圈呼叫 pthread_create() 啟動五個 random_points() 執行緒；每個執行緒以 srand(time(0)) 設定亂數種子，然後在 1000 次迭代中隨機產生落在 $[-1,1]\times[-1,1]$ 的點，若落入單位圓（$x^2+y^2\le 1$）則以 pthread_mutex_lock()／unlock() 保護遞增 counter，最後呼叫 pthread_exit() 結束。主執行緒透過五次 pthread_join() 等待所有工作執行緒結束，釋放 tids 記憶體，並以公式計算出 π 的估算值，最後以 printf 輸出結果。

3. 程式輸出

```
xavier1021@LAPTOP-68NFQUBM:/mnt/d/opration system/os_hw3$ gcc hw3.c -o hw3_output
xavier1021@LAPTOP-68NFQUBM:/mnt/d/opration system/os_hw3$ ./hw3_output
Estimated value of pi: 3.135200
```