Title: NTU IM Operating-System HW 03
Student ID: R12631070
Name: 林育新

# 問答題 (6.14)

6.14 題目解答：

**a. 請說明此程式碼中有哪些競爭條件（race condition）？**

`allocate_process()` 和 `release_process()` 對共享變數 `number_of_processes` 存在競爭條件。

---

**b. 假設有一個名為 `mutex` 的互斥鎖，並提供 `acquire()` 與 `release()` 兩個操作。請指出在程式中應該在哪些地方加上鎖來防止競爭條件。**

```
int allocate_process() {
    int new_pid;

    acquire(mutex);  // 🔒 鎖定互斥鎖

    if (number_of_processes == MAX_PROCESSES) {
        release(mutex);  // 🔓 解鎖
        return -1;
    } else {
        /* 修改共享資源 */
        ++number_of_processes;
    }

    release(mutex);  // 🔓 解鎖

    return new_pid;
}

void release_process() {
    acquire(mutex);  // 🔒 鎖定互斥鎖

    /* 修改共享資源 */
    --number_of_processes;

    release(mutex);  // 🔓 解鎖
}
```

---

# 程式題 hw3.c (6.33)

題目: 蒙特卡羅方法估算圓周率 π (使用 Pthreads)

1. **使用方式**

```
Make
./hw3
```

2. **算法解釋：**
   蒙特卡羅方法: 用大量隨機資料來模擬問題的解答，並從中估算出結果。
   假設有一個半徑為1的圓(面積為π)，內接於一個邊長為2的正方形(面積為4)中。
   隨機產生大量的點落在這個正方形內，統計有多少點落在圓內。
   估計的π值：π ≈ 4 × (圓內點數量) / (總點數量)； 概念相當於 正方形面積 * (圓形面積/正方形面積)。

3. **子線程邏輯：**

   - 生成隨機點並計算圓內點數，結束前用mutex更新global的point數量。

```c
void *generate_points(void *arg)
{
    const int POINTS_PER_THREAD = 1000; // 每個線程固定生成1000個點
    unsigned long long local_count = 0; // 本地計數器

    // 初始化隨機數生成器 (使用線程ID作為種子)
    unsigned int seed = time(NULL) ^ pthread_self();

    for (int i = 0; i < POINTS_PER_THREAD; ++i)
    {
        // 生成[-1, 1]範圍內的隨機數
        double x = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;
        double y = (double)rand_r(&seed) / RAND_MAX * 2.0 - 1.0;

        // 檢查點是否在單位圓內 (x² + y² ≤ 1)
        if (x * x + y * y <= 1.0)
        {
            local_count++; // 如果在圓內，本地計數加1
        }
    }

    // 使用互斥鎖安全地更新全域計數
    pthread_mutex_lock(&mutex);      // 上鎖
    points_in_circle += local_count; // 更新圓內點數
    pthread_mutex_unlock(&mutex);    // 解鎖

    return NULL;
}
```

4. **主線程邏輯：**

   - pthread_create 創建線程。
   - pthread_join 等待所有子線程完成以後，計算π的估計值。

```
int main()
{
    const int NUM_THREADS = 5;       // 固定創建5個線程
    pthread_t threads[NUM_THREADS]; // 線程陣列

    // 初始化隨機數種子 (僅用於主線程)
    srand(time(NULL));

    // 創建並啟動所有線程
    for (int i = 0; i < NUM_THREADS; ++i)
    {
        pthread_create(&threads[i], NULL, generate_points, NULL);
    }

    // 等待所有線程完成
    for (int i = 0; i < NUM_THREADS; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    // 計算並輸出π的估算值
    double pi_estimate = 4.0 * points_in_circle / (NUM_THREADS *
1000.0);
    printf("估算的π值: %.15f\n", pi_estimate);
    printf("總點數: %d\n", NUM_THREADS * 1000);
    printf("圓內點數: %llu\n", points_in_circle);

    // 銷毀互斥鎖
    pthread_mutex_destroy(&mutex);

    return 0;
}
```