# Assignment9


## Team18

112062519 廖思愷

112062636 游竣量

111065547 游述宇


- **Explanations of implemented code**

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 5
```

匯入必要的 header file 與定義常數 NUM_THREADS：

- pthread.h 是 POSIX 線程庫的頭文件，用於多線程程式設計
- stdio.h 和 stdlib.h 分別用於標準輸入輸出和一般實用工具的功能
- NUM_THREADS 定義了將要創建的線程數量，這裡設為 5

```c
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int count = 0;
int flag = 0;
```

- 定義兩個用於同步的 primitives：一個互斥鎖（mutex）和一個條件變量（cond）
- 互斥鎖用於控制對共享資源的訪問，以防止 race condition
- 條件變量用於線程間的信號通訊
- count 用於追蹤已達到 barrier 的線程數量
- flag 用於指示所有線程是否都已到達 barrier

```c
void pthread_barrier_wait_() {
  pthread_mutex_lock(&mutex);
  count++;
  if (count == NUM_THREADS) {
    flag = 1;
    pthread_cond_broadcast(&cond);
  } else {
    while (!flag) {
      pthread_cond_wait(&cond, &mutex);
    }
  }
  pthread_mutex_unlock(&mutex);
}
```

- pthread_barrier_wait_ 函數模擬一個 barrier 的行為
- 當一個線程調用這個函數時，它會先鎖定互斥鎖，然後增加 count
- 如果 count 達到了線程總數（即所有線程都已達到 barrier），它會設置 flag 並通過條件變量喚醒所有等待的線程
- 如果 count 尚未達到線程總數，線程將進入等待狀態
- 在完成這些操作後，互斥鎖被釋放

```c
void *do_work(void *arg) {
  pthread_barrier_wait_();
  pthread_t thread_id = pthread_self();
  printf("Thread %lu running\n", (unsigned long)thread_id);

  return NULL;
}
```

- do_work 是每個線程執行的函數
- 它首先調用 pthread_barrier_wait_ 以等待其他線程，一旦所有線程都到達障礙，它將獲取自身的線程 ID 並打印一條消息，指示該線程正在運行

main function：

```c
pthread_t threads[NUM_THREADS];
```

- 定義一個 pthread_t 類型的陣列，名為 threads，其大小由 NUM_THREADS 定義
- 這個陣列將用於儲存線程的 ID

```c
for (int i = 0; i < NUM_THREADS; i++) {
  printf("Starting thread %d\n", i);
  if (pthread_create(&threads[i], NULL, do_work, NULL) != 0) {
    perror("pthread_create error");
    exit(1);
  }
}
```

創建 NUM_THREADS 數量的線程：

- 對於每個即將創建的線程，打印一條消息（"Starting thread i"）
- pthread_create 函數用於創建新線程，這個函數接受幾個參數：
  - &threads[i]：一個指向線程 ID 的指針
  - NULL：指定線程的預設屬性
  - do_work：線程將要執行的函數
  - NULL：傳遞給 do_work 函數的參數，這裡沒有使用
- 錯誤處理：如果 pthread_create 返回非零值，表示線程創建失敗，這時會打印錯誤信息並退出程序

```c
for (int i = 0; i < NUM_THREADS; i++) {
  pthread_join(threads[i], NULL);
}
```

- 在這個迴圈中，pthread_join 函數被用於等待每個線程完成其執行，這確保了主線程（執行 main 函數的線程）會等待所有子線程完成它們的任務才繼續執行。

```c
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&cond);
```

清理資源：

- pthread_mutex_destroy 函數銷毀了先前初始化的互斥鎖
- pthread_cond_destroy 函數銷毀了條件變量

- **screenshot of result**