# CS305 作業系統概論

# Prog. #2 Multithreading 說明報告

汪文豪(學號：1071710)

## ➢ 如何編譯與測試操作程式：

1. 編譯：g++ 檔名.cpp –o 檔名 –lrt -pthread
2. 執行：./檔名 欲讀入資料檔案之檔名

   補充：資料檔案內之 infix 必須無空格，否則會報錯

   只能有一行運算式，程式只會執行第一行指令
3. 輸出結果

範例(資料檔案檔名為 prog2data)：

一. g++ hw2.cpp –o prog2 –lrt –pthread

二. ./prog2 prog2data

三. 輸出結果

```
ryan@ubuntu:~/Desktop$ g++ hw2_cop.cpp -o prog2 -lrt -pthread
```

```
ryan@ubuntu:~/Desktop$ ./prog2 prog2data
The infix input: (1+2*4+((7-5))/2)
[Child 1]+ + 1 * 2 4 / - 7 5 2 = 10 216ms
[Child 2]1 2 4 * + 7 5 - 2 / + = 10 25ms
[Child 1 tid=26512] 216ms
[Child 2 tid=26513] 25ms
[Main thread] 716ms
ryan@ubuntu:~/Desktop$ ./prog2 prog2data
The infix input: (1+2*4+)((7-5))/2)
[Child 2]syntax error 20ms
[Child 1]syntax error 4ms
[Child 1 tid=26518] 4ms
[Child 2 tid=26519] 20ms
[Main thread] 369ms
ryan@ubuntu:~/Desktop$ ./prog2 prog2data
```

若要測試程式可直接從資料檔案內改資料，儲存後直接執行程式即可

## ➢ 設計理念：

**本次使用到的特殊函式庫**

```
1   #include<stdlib.h>
2   #include<stdio.h>
3   #include<time.h> // about system wall-clock time
4   #include<fstream> // file
5   #include<string> // string structure
6   #include<iostream>
7   #include<pthread.h> // pthread API
8   #include<unistd.h> // get tid
9   #include<stack> // stack structure
10  #include<vector> // vector structure
11  #include <cmath>
```

1.與 2.是 C/C++語言標準函式庫

3.time.h 是計算 wall-clock time

4.是為了做讀檔，會用到 ifstream

7.是因應本次作業使用的 pthread API

8.為了取得其 thread 之 tid，使用的函式庫

11.是為了做四捨五入所用的

其餘皆是其作業所需的基本資料結構所用到的

**本次程式說明：**

✓ 完成部分：

**基礎 1：主執行緒從命令列讀入檔名，並完成內容輸出。本項滿分 10 分**

391 行至 396 行宣告接收資料之變數並讀入資料內容，存入 statement 之 string 結構中，將 infix 內容做輸出。

```
387 int main(int argc,char* argv[])
388 {
389     clock_t start, end;
390     start = clock(); // start computing time
391     ifstream fin;
392     fin.open(argv[1]);
393     /* read files*/
394     string statement;
395     getline(fin,statement);
396     cout<<"The infix input: "<<statement<<"\n";
397     pthread_t work_thread[2];
398     parameter args[2];
399     /*get infix*/
```

**基礎 2：主執行緒能正確產生子執行緒。本項滿分 10 分。**

透過陣列產生兩個子執行緒，倘若創建不成功，406 行會判斷到且將錯誤印出

```
397     pthread_t work_thread[2];
398     parameter args[2];
399     /*get infix*/
400     /*set work thread*/
401     for(int i=0;i<2;i++)
402     {
403         args[i].number=i;
404         args[i].statement = statement;
405         if(pthread_create(&work_thread[i],NULL,work_computing,(void*)&args[i])!=0){
406             cerr<<"error occurred by work"<<i<<"\n";
407         };
408     }
409     for (int i = 0 ;i<2;i++)
410     {
411         pthread_join(work_thread[i], NULL);
412     }
413     for (int i = 0 ;i<2;i++){
414     cout<<"[Child"<<" "<<i+1<<" "<<"tid="<<args[i].tid<<"] "<<args[i].time<<"ms\n";
415     }
416     end = clock();// end computing time
417     double diff = (double)(end - start);
418     cout<<"[Main thread] "<<diff<<"ms\n";// 1 seconds = 1000 ms
419     return 0;
420 };
421
422
```

**基礎 3：在不考慮大數運算下，子執行緒能夠正確完成正確的 prefix 與 postfix 的轉換與運算，並印出結果。本項滿分 40 分。**

336 行之 work_computing function 即是處理 pthread 之運算，由於兩個 thread 之先後順序無法控制，有可能導致重疊輸出，故一開始先用 mutex_lock，之後做時間的計算，先取得開始執行的時間(339 行)，要離開前再計算其結束時間(377-378 行)並做兩者間的時間差並印出與儲存至 struct，之後再解鎖並結束 thread 之執行。

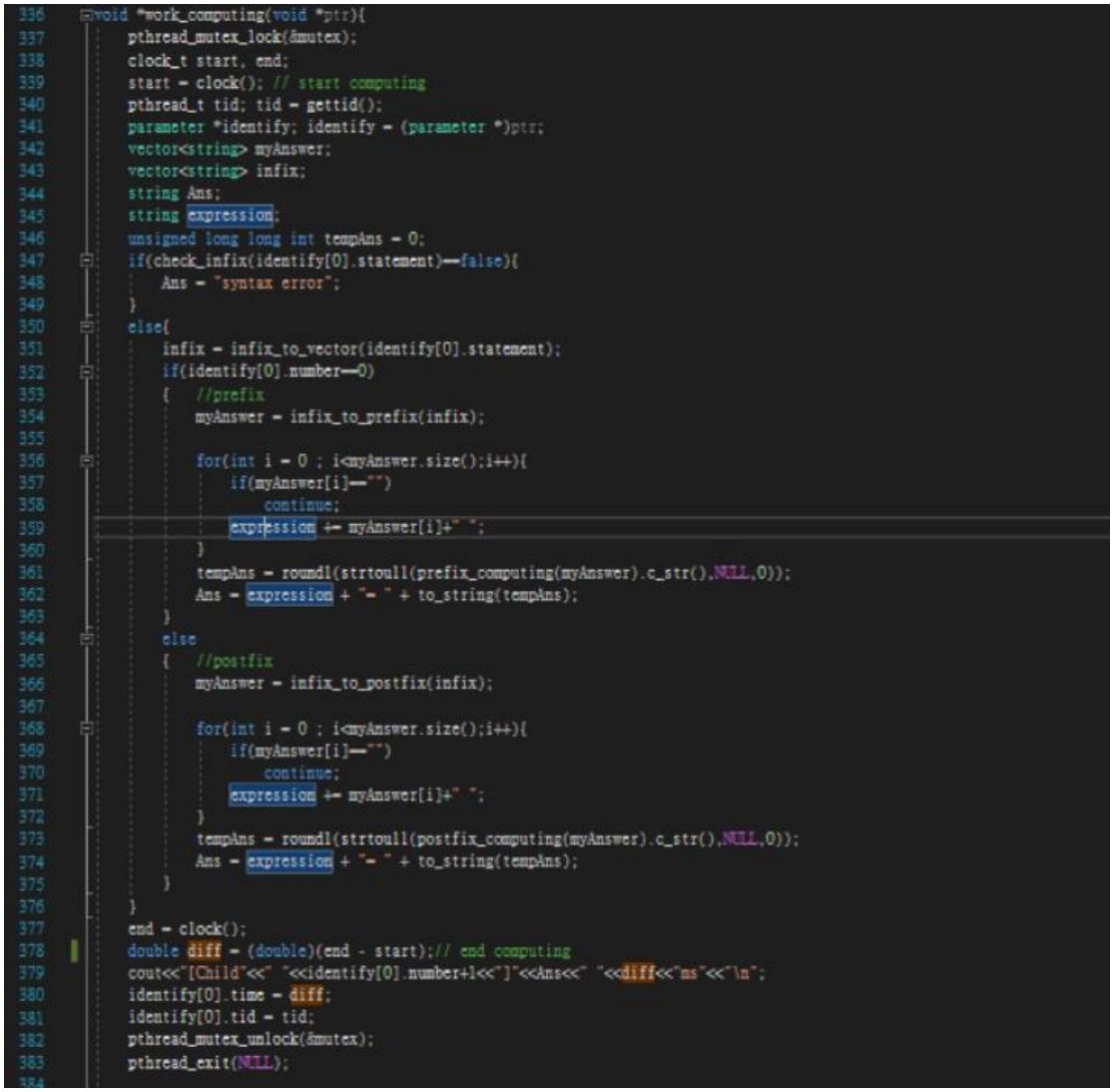
341 行則是取得傳入進來之多參數 struct 資料

347 判斷 infix 是否合法，若不合法就跳過運算式顯示 syntax error

350 行開始運算之 block，首先之 351 行則是將 string 轉成 vector，將 operand、operator、parentness 等切成一塊一塊的 block

354 行則是先將 infix 轉成 prefix，之後將其表達式儲存到 expression 中，由於 operand 和 operator 要分隔，因此後面會再加一個空格分隔

Ans 則是將 expression 與等號和使用 prefix_computing 計算其結果答案，全數存至 Ans 中，而 post 與 prefix 一樣，只是換另一個 function。另外 tempAns 則是因為需要四捨五入取整數，故在計算的時候是使用 long double，之後使用 roundl 將 long double 四捨五入取整數，放置 unsigned long long int 之 tempAns 變數中。此外判斷是 prefix or postfix 全看當初放入筆者所設計之 parameter strcut 之 number 來做區分，辨別哪一個是 work_thread1 或 2，並進入自己的計算區塊中。最後計算出的答案統一於 379 行印出，並將其時間與 tid 儲存到其傳進來的 struct 之 identify 中，最後統一將其時間與 tid 傳回 main thread 印出。



```
336  void *work_computing(void *ptr){
337      pthread_mutex_lock(&mutex);
338      clock_t start, end;
339      start = clock(); // start computing
340      pthread_t tid; tid = gettid();
341      parameter *identify; identify = (parameter *)ptr;
342      vector<string> myAnswer;
343      vector<string> infix;
344      string Ans;
345      string expression;
346      unsigned long long int tempAns = 0;
347      if(check_infix(identify[0].statement)==false){
348          Ans = "syntax error";
349      }
350      else{
351          infix = infix_to_vector(identify[0].statement);
352          if(identify[0].number==0)
353          {   //prefix
354              myAnswer = infix_to_prefix(infix);
355
356              for(int i = 0 ; i<myAnswer.size();i++){
357                  if(myAnswer[i]=="")
358                      continue;
359                  expression += myAnswer[i]+" ";
360              }
361              tempAns = roundl(strtoull(prefix_computing(myAnswer).c_str(),NULL,0));
362              Ans = expression + "= " + to_string(tempAns);
363          }
364          else
365          {   //postfix
366              myAnswer = infix_to_postfix(infix);
367
368              for(int i = 0 ; i<myAnswer.size();i++){
369                  if(myAnswer[i]=="")
370                      continue;
371                  expression += myAnswer[i]+" ";
372              }
373              tempAns = roundl(strtoull(postfix_computing(myAnswer).c_str(),NULL,0));
374              Ans = expression + "= " + to_string(tempAns);
375          }
376      }
377      end = clock();
378      double diff = (double)(end - start);// end computing
379      cout<<"[Child"<<" "<<identify[0].number+1<<"]"<<Ans<<" "<<diff<<"ms"<<"\n";
380      identify[0].time = diff;
381      identify[0].tid = tid;
382      pthread_mutex_unlock(&mutex);
383      pthread_exit(NULL);
```

**基礎 4：在不考慮大數運算下，子執行緒能夠正確處理 infix 運算式中的各種錯誤。例如“1+2+”，應輸出錯誤訊息。本項滿分 20 分。**

如上一題所述，在 347 行會做判斷，若有問題即會將 syntax error 放入 ans 並顯示：

```
ryan@ubuntu:~/Desktop$ ./prog2 prog2data
The infix input: 1+2+
[Child 2]syntax error 4ms
[Child 1]syntax error 4ms
[Child 1 tid=26735] 4ms
[Child 2 tid=26736] 4ms
[Main thread] 543ms
```

而檢查 infix 之錯誤寫在第 260 行之 check_infix function 中，由於程式碼太長故不貼上。最主要的核心概念是計算必須按照 operand + operator + operand 之順序排列，因此最後的數量會是 operand_number == operator_number + 1，若否則是錯誤；過程中也會判斷是否按照 operand + operator + operand 之順序，否則也會是錯誤。而左括號 – 右括號數量必須衡大於 0，否則代表右括號比左括號先出現，直接 return false

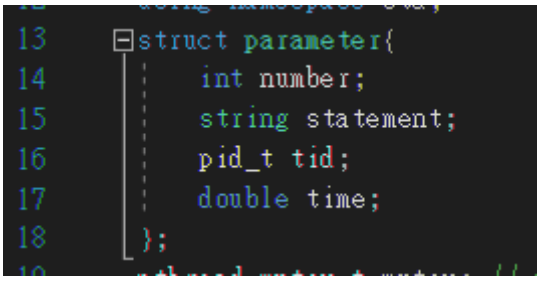
另外還有一些括號的額外判斷也都有做，但較為複雜故不在此贅述

**基礎 5：主執行緒與子執行緒不透過任何 IPC 機制交換資料，並且正確印出子執行緒的 tid 與執行時間。本項滿分 20 分。**

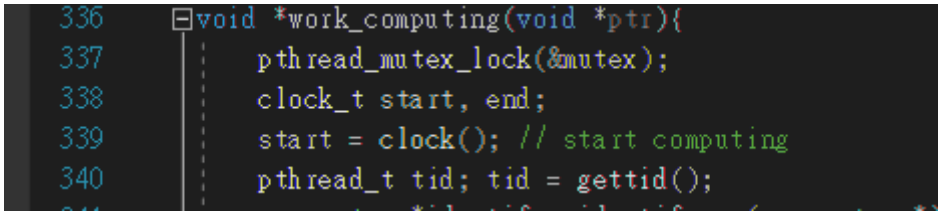
本次程式透過 struct 傳遞與操作其多參數，儲存 time、tid 和 infix 表達式與判斷是為 work_thread1 或 work_thread2 之 number 共四個參數

時間也都分別在子執行緒開始執行將時間存入 start，即將結束時存入 end，並計算之間的差後印出並將資料存入其 struct paramete 指標所指向的位址，之後在 main thread 印出

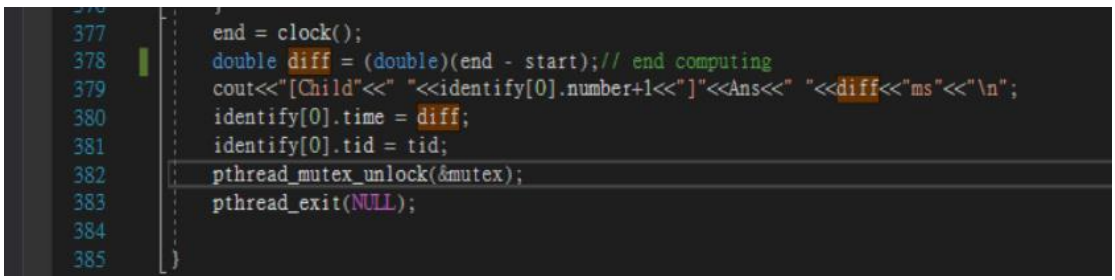
tid 則是透過 gettid()取得，資料交換方式相同




取得 tid




取得開始時間和 tid




存入 parameter 指標指向之位址

**其他 Function 用途解說：**

1. 20 行之 order_list，postfix & prefix 需要用到，判斷其 operator 優先序
2. 29 行之 check_isOperator，判斷字元是否是 operator
3. 34 行之 check_stringOperator，因為之後會將表達式做切割，故不會是字元而是字串，為方便操作則多寫一個判斷字串是否是 operator 之 function
4. 39 行之 check_isNumber，判斷其字元是否是 0 – 9 範圍的數字字元
5. 45 行之 infix_to_vector，將 infix 切成一塊一塊的 block，方便做計算
6. 80 行之 infix_to_postfix，將 infix 轉成 postfix 表達式
7. 122 行之 reverse_infix，由於 prefix 需要將 infix 倒置，故寫此 function
8. 130 行之 infix_to_prefix，將 infix 轉成 prefix 表達式
9. 187 行之 compute，因應其 prefix & postfix 計算，會需要用到這個 function，先轉成 long double 計算，以利之後完整結果做四捨五入，將兩個 operand 計算後將結果轉成 string 回傳
10. 209 和 234 行之 postfix computing & prefix computing 則是做其表達式的運算