**CS305作業系統概論 Prog. #2 Multithreading. 說明報告.docx**

**作者　：1043335 賴詩雨**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **修改日期** | **說明** | **修改者** |
| **1.0.0.0** | **2017-05-08** | **初稿** | **賴詩雨** |
| **1.0.0.1** | **2017-05-11** | **修改** | **賴詩雨** |
|  |  |  |  |
|  |  |  |  |

**目錄**

[第1章 程式完成部分: 2](#_Toc482228753)

[第2章 設計理念 2](#_Toc482228754)

[2.1 程式中特殊介紹 2](#_Toc482228755)

[2.2 Thread產生示意圖 4](#_Toc482228756)

[第3章 程式流程簡介 5](#_Toc482228757)

[第4章 程式輸出簡介 6](#_Toc482228758)

[第5章 程式如何編譯 8](#_Toc482228759)

[第6章 如何操作 8](#_Toc482228760)

1. 程式完成部分:
2. 基本功能
3. 進階功能– advmerge
4. 進階功能– advquick
5. 設計理念

利用pthread的共用變數讓所有work\_thread等待，直到main\_thread讀好資料後，啟動變數更改成1(Ready)，所有的work\_thread才開始一起排序。

* 1. 程式中特殊介紹

1. 有三個結構，分別為 \_S\_GLOBAL 、 \_S\_THREAD 和 \_S\_THREAD
2. **typedef** struct \_S\_GLOBAL

**{**

int Ready**;**

int sleepTime**;**

int Value**[**length**];**

int time**[**5**][**2**];**

int tid[5];

**}** S\_GLOBAL**,** **\***P\_GLOBAL**;**

1. **typedef** struct \_S\_THREAD

**{**

int ID**;**

int MyValue**[**length**];**

P\_GLOBAL SG**;**

**}** S\_THREAD**,** **\***P\_THREAD**;**

1. **typedef** struct \_S\_ADV

**{**

int TID**;**

int CID**;**

int HalfData**[**length**];**

P\_THREAD ST**;**

**}** S\_ADV**,** **\***P\_ADV**;**

1. 整個程式中有用到的結構宣告為

P\_THREAD pT**[**5**];** // 用於基本功能中

P\_ADV M**[**2**];** // 用於進階功能 - advmerge

P\_ADV Q**[**2**];**  // 用於進階功能 - advquick

1. \_S\_GLOBAL

ReadysleepTime Value**[**length**]** time**[**5**][**2**]** tid[5]

IDMyValue**[**length**]**

SG

IDMyValue**[**length**]**

SG

IDMyValue**[**length**]**

SG

……

\_S\_THREAD pT**[**0**]** pT**[**1**]** pT**[**2**]** pT**[**3**]** pT**[**4**]**

1. 第六點的結構為pT[0]~pT[4]的結構，淺藍色為共用變數，灰色為區域變數
2. \_S\_GLOBAL

ReadysleepTime Value**[**length**]** time**[**5**][**2**]** tid[5]

ID

MyValue**[**length**]**

SG

ID

MyValue**[**length**]**

SG

此區為Q[0],Q[1]的公用變數區域

* pT[4]

此區為M[0],M[1]的公用變數區域

* pT[3]

TID

CIDHalfData**[**length**]**

TG

TID

CIDHalfData**[**length**]**

TG

TID

CIDHalfData**[**length**]**

TG

TID

CIDHalfData**[**length**]**

TG

\_S\_ADV M**[**0**]** M**[**1**]** Q**[**0**]** Q**[**1**]**

1. 第八點的結構為 M[0]、M[1]、Q[0]、Q[1] 的結構，黃色區域為進階功能中再次創造的pThread的區域變數，灰色區域為M[0]、M[1]的共用變數區和Q[0]、Q[1]的共用變數區，淺藍色為大家的共用變數區
   1. Thread產生示意圖

**EX :**

**pthread\_create (&KidThread1, NULL, (void \*) &work\_thread, pT[0]);**

Main\_thread

主程式

M [1]

M [0]

Q [0]

Q [1]

**work\_thread()**

**advQ ()**

**advM ()**

**advQ\_work ()**

**advM\_work ()**

1. 程式流程簡介
2. 在main\_thread設定好結構的基本需求後，會開始crate五個work\_thread ( pT[0]~pT[4] ) ，並會在讀入資料後等待他們結束。
3. 在所有work\_thread建立好之後，他們才會收到訊號一起往下執行。
4. pT[0]~p[2]會如 p.4 的圖一樣，呼叫 work\_thread()，然後會去檢查main\_thread讀入資料沒，若還沒，則會根據在命令列輸入的秒數睡覺及印出現在時間。
5. 再來等到main\_thread讀好資料並改變啟動變數後(Ready)，pT[0]~pT[2]會一起繼續往下執行並根據自己的ID去做他們要做的排序，並且輸出排序的檔案和將 (1) 回傳時間 及 (2)排序時間放到共用變數中。
6. pT[3]是要執行advMerge的，他會如p.4的圖一樣，呼叫 advM()，並在advM()中設定好結構的基本需求後，create 兩個thread，並呼叫advM\_work()，advM()則會等待advM\_work()結束。
7. 在advM\_work()中，會有M[0]和M[1]，他們一樣會等到main\_thread讀好資料後才一起繼續執行，然後根據自己的CID去決定他們的分工排序，排序完之後再回到advM()去做最後排序並輸出檔案，以及將 (1) 回傳時間 及 (2)排序時間 放到共用變數中。
8. pT[4]是要執行advQuick的，他會如p.4的圖一樣，呼叫 advQ()，並在advQ()中設定好結構的基本需求後，create 兩個thread，並呼叫advQ\_work()，advQ()則會等待advQ\_work()結束。
9. 在advQ\_work()中，會有Q[0]和Q[1]，他們一樣會等到main\_thread讀好資料後才一起繼續執行，然後根據自己的CID去決定他們的分工排序，排序完之後再回到advQ()去輸出檔案，以及將 (1) 回傳時間 及 (2)排序時間 放到共用變數中。
10. 五個work\_thread都執行結束後，main\_thread才可以繼續執行，他會根據thread們放入共用變數區的時間，去做排序，再把時間印出。

Note: 如果時間相同，則按照回傳時間的順序印出。如果回傳時間相同，以bubble sort, merge sort 以及quick sort的順序印出。

1. 程式輸出簡介

**黑字 => 會印出的訊息**

**藍字 => 程式的動作**

**紅字 => 平行執行的部分**

**---------------------------------------------------------------------**

Main thread : work thread for bubblesort

Main thread : work thread for mergesort

Main thread : work thread for quicksort

Main thread : work thread for advMerge

Main thread : work thread for advQuick

advMerge : work thread for child\_advmerge\_1

advMerge : work thread for child\_advmerge\_2

advQuick : work thread for child\_advquick\_1

advQuick : work thread for child\_advquick\_2

**以創建的work\_thread在等待main\_thread create 完畢**

Bubblesort : [ 現在時間 ] sleep 3 second

mergesort : [現在時間] sleep 3 second

quicksort : [現在時間] sleep 3 second

advMerge : [現在時間] sleep 3 second

advQuick : [現在時間] sleep 3 second

child\_advmerge\_1 : [ 現在時間 ] sleep n second

child\_advmerge\_2 : [ 現在時間 ] sleep n second

child\_advquick\_1 : [ 現在時間 ] sleep n second

child\_advquick\_2 : [ 現在時間 ] sleep n second

**所有的work\_thread在等待main\_thread 讀入資料且根據輸入數字去睡覺**

Main thread : read data

Main thread : write data to the shared area

Main thread : tell work thread start to work

**所有wotk\_thread開始執行，main\_thread等待他們執行結束**

**>> pT[0] ~ pT[2]**

Work\_thread : sort begin

Work\_thread : sort end

Work\_thread : sort 秒數

Work\_thread : sort 寫入 txt

Work\_thread : 將時間放到與main\_thread的共用區域

**>> pT[3] avdmerge**

pT[3] : create M[0]

pT[3] : create M[1]

**create出的兩個 thread等待 main\_thread 讀入資料並更改啟動變數**

M\_thread : sort begin

M\_thread : sort end

M\_thread : 將結果放到與 pt[3]的共用區域

pT[3] : advmerge begin

pT[3] : advmerge end

pT[3] : advmerge 秒數

pT[3] : advmerge 寫入 txt

pT[3] : 將時間放到與main\_thread的共用區域

**>> pT[4] avdquick**

pT[4] : create Q[0]

pT[4] : create Q[1]

**create出的兩個 thread等待 main\_thread 讀入資料並更改啟動變數**

Q\_thread : sort begin

Q\_thread : sort end

Q\_thread : 將結果放到與 pt[4]的共用區域

pT[4] : advquick 秒數

pT[4] : advquick 寫入 txt

pT[4] : 將時間放到與main\_thread的共用區域

Main thread : 將時間排序並由小到大印出 ( 共有五個 )

1. 程式如何編譯
2. 有附上Makefile檔提供操作

OR

1. 在命令列輸入 : gcc 1043335\_02.c -lpthread -o 1043335\_02
2. 編譯成功後，輸入 : ./1043335\_02 number.txt 3

**NOTE : number.txt 可改成要讀取的檔案名稱 && 3 可改成 1~5任意數**

1. 如何操作
2. 將 Makefile 中 run 的number.txt 改成要讀的檔案名稱將 number.txt 改成要讀的檔案名稱，且數字 3 改成想檢查的秒數區間
3. make main => 編譯
4. make run => 執行
5. make start => 編譯 & 執行
6. make clean => 清除編譯、執行後產生的檔案
7. make all => 清除檔案後編譯執行
8. 或是可以直接在命令列輸入 **第五章** 提供的指令