# CS305 作業系統概論 Prog. #1 Proc. 說明報告

1043335 賴詩雨

## 一、程式完成部分:

a. 基本功能

b. 進階功能 - advmerge

c. 進階功能 - advquick

## 二、設計理念

利用 shared memory 讓所有 child 等待，直到 parent 讀好資料後，放入開始指令（ T ），child 才開始一起排序。

## 三、程式中特殊介紹

**NOTE: 進階功能中再次 fork()出來的子程式稱為 grandchild，child 皆為 main（ parent ）fork() 出的（ 請參考下夜的圖 ）**

1. 有兩個結構，分別為 struct _S_CHILD 和 struct _S_MAIN

2. typedef struct _S_CHILD

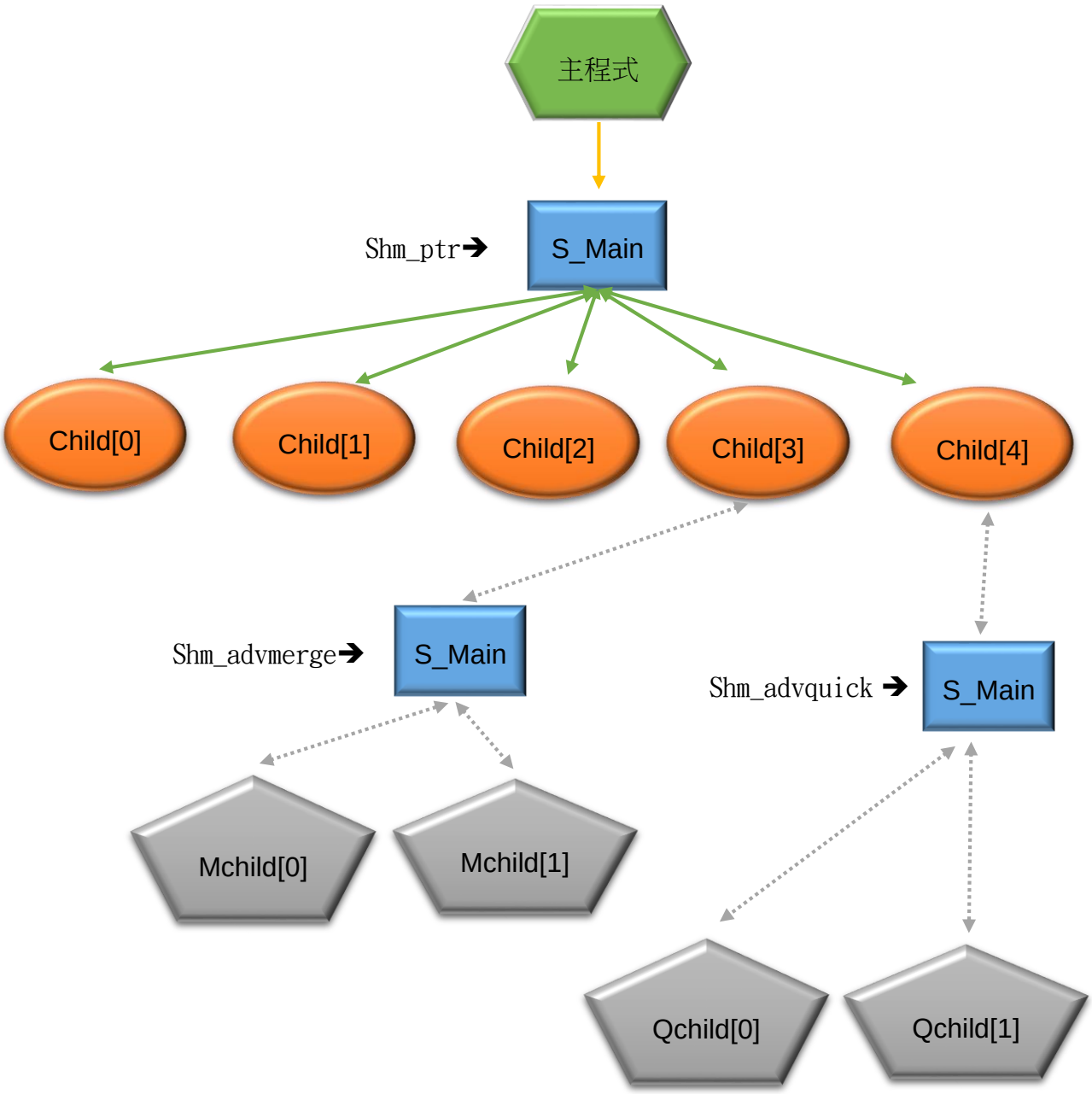
```
{
  int      Done;          // 程式是否完成
  int      ResponseTime;  // 程式執行時間
} S_CHILD, *P_CHILD;
```

3. typedef struct _S_MAIN

```
{
  char     Ready;         // 通知 child 一起開始的訊號
  int      Value[length];// 放入從檔案讀取的 10000 個數字
  S_CHILD  A;             // 放平行執行中的第一個 child 的資料
                          以及進階功能中的 child 的資料
  S_CHILD  B;             // 放平行執行中的第二個 child 的資料
                          以及進階功能中 grandchild1 的資料
  S_CHILD  C;             // 放平行執行中的第三個 child 的資料
                          以及進階功能中 grandchild2 的資料
} S_MAIN, *P_MAIN;
```

4. P_MAIN Shm_ptr, Shm_advmerge, Shm_advquick，分別讓三個 shared memory 有上述我建立的結構
5. Shm_ptr，用於基本功能中 parent 和三個子程式的資料傳遞
6. Shm_advmerge，用於進階功能中 child 和 grandchild 的資料傳遞
7. Shm_advquick，用於進階功能中 child 和 grandchild 的資料傳遞

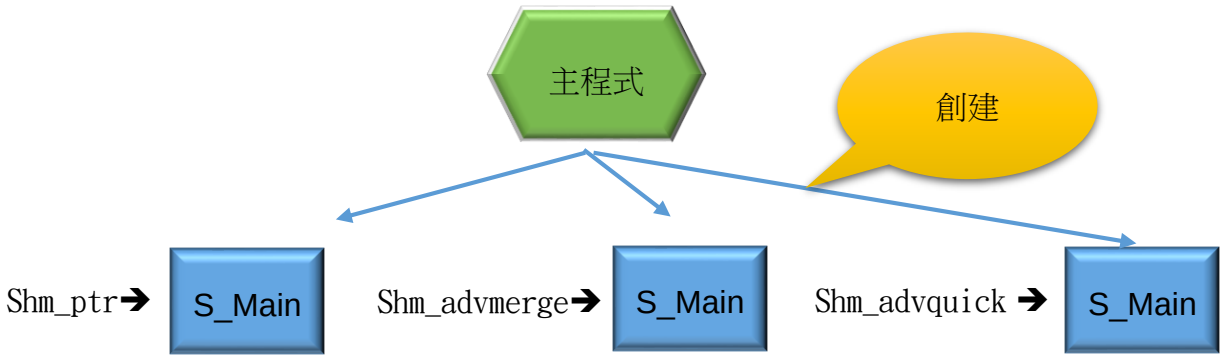
< 灰底為 grandchild >



## 四、程式流程簡介 ：

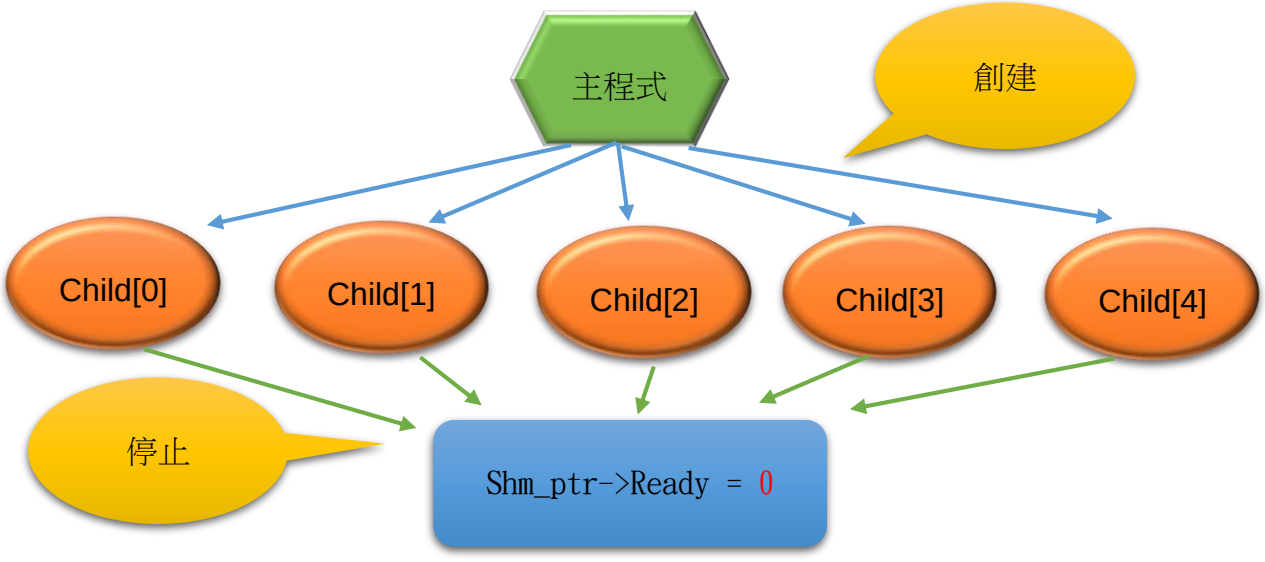
1. parent 會先取得自己的 pid ( 取名為 self ) 。
2. 有一個陣列 child[5] 去放 fork() 出來的，其中：
   childpid ➔ child[0]~ child[2] 為基本功能的三個平行的子程式。
   child[3]為 advmerge，child[4]為 advquick。
3. 接下來 Parent 創建 shared memory，總共有三個 ( Shm_ptr, Shm_advmerge, Shm_advquick )。
   Shm_ptr 平行三個子程式 ( child[0]~ child[2] ) 和 parent 共用。
   另外兩個 Shm_advmerge 和 Shm_advquick 分別為兩個進階功能 ( child[3]，child[4] )，和 Mchild 間傳遞資料的 shared memory。
   三個 shared memory 都在一開始會建立好。

   NOTE ： 三個 shared memory 均相同，內容為我設計的 structure




4. 創建完 shared memory 之後，會在 parent 裡面 fork() 出 5 個 child
5. 這五個 child 會開始讀取 shared memory (Shm_ptr)的 ready *(一開始都為 0)*，並被卡在這裡，直到 ready 被 parent 放入 T，所有 child 才會繼續往下動作

   NOTE ： 這裡是為了讓 child 平行執行



6. 再來，parent 會把檔案資料放入 shared memory，並把 T 放入 ready，讓五個 child 能繼續執行，Parent 利用 wait 去等待 child
7. 平行執行的三個子程式，會去根據我給的編號（ child[0]，child[0]，child[2] 對應 S_CHILD A，S_CHILD B，S_CHILD C ）做各自的排序，並將執行時間放回到 shared memory
8. child[3] 會 fork()兩個他自己的 Mchild，讓兩個 Mchild 平行運作，各自處理 5000 個資料，最後放入 advmerge 的 shared memory 給 child[3] 做最後合併，並把時間放到此 shared memory 中，回傳給 parent
9. child[4] fork()兩個他的 Qchild，讓兩個 Qchild 平行運作，各自處理第一個 pivot 分好的資料，最後放入 advquick 的 shared memory 給 child[4](此時等於排序完畢)，child[4] 將時間放到此 shared memory 中，回傳給 parent

   NOTE ： 因為 advmerge 和 advquick 速度太快，因此有加印出 ns
10. Parent 收到五個 child 結束的訊息後，會讀取他們的回傳時間，再做排序，由小到大印出

**< 在執行程式的過程中，程式會印出他們目前的動作，請參考下頁 >**

## 五、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

---

Parent : fork for bubblesort

Parent : fork for quicksort

Parent : fork for mergesort

Parent : fork for advmerge

Parent : fork for advquick

fork 出的所有 child 在等待 Parnet 放入 T

Parent : read data to shared memory

Parent : tell childs to read data

fork 出的所有 child 執行，Parent wait childs

>> child[0] ~ child[3]

child : sort begin

child : sort end

child : sort 秒數

child : sort 寫入 txt

>> child[3] avdmerge

child[3] : fork Mchild[0]

child[3] : fork Mchild[1]

fork 出的兩個 Mchild 等待 child[3] 放入 T

child[3] 放入 T 並 wait

Mchild : sort begin

Mchild : sort end

child[3] : advmerge begin

child[3] : advmerge end

child[3] : advmerge 秒數 ( ms & ns )

child[3] : advmerge 寫入 txt

>> child[4] avdquick

child[4] : fork Qchild[0]

child[4] : fork Qchild[1]

fork 出的兩個 Qchild 等待 child[4] 放入 T

child[4] 放入 T 並 wait

Qchild : sort begin

Qchild : sort end

child[4] 拿到兩個 Qchild 排好的資料不須重新排過

child[4] : advquick 秒數 ( ms & ns )

child[4] : advquick 寫入 txt

Parent : the execution time rank

Parent : 時間由小到大印出 ( 共有五個 )

## 六、程式如何編譯

1. 有附上 Makefile 檔提供操作

   OR

2. 在命令列輸入　　: gcc -o 1043335_01 1043335_01.c -lrt
3. 編譯成功後，輸入 : ./1043335_01 number.txt

NOTE : number.txt 可改成要讀取的檔案名稱

## 七、如何操作

1. 將 Makefile 中 start 的 number.txt 改成要讀的檔案名稱將 number.txt 改成要讀的檔案名稱
2. make main  => 編譯
3. make run  => 執行
4. make start => 編譯 & 執行
5. make clean => 清除編譯、執行後產生的檔案
6. make all  => 清除檔案後編譯執行
7. 或是可以直接在命令列輸入 **六** 提供的指令