CS305 作業系統概論 Prog. #1 Proc. 說明報告

1043335 賴詩雨

## **程式完成部分:**

1. 基本功能
2. 進階功能– advmerge
3. 進階功能– advquick

## **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。

## **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;

1. 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;

1. P\_MAIN Shm\_ptr, Shm\_advmerge, Shm\_advquick，分別讓三個shared memory有上述我建立的結構
2. Shm\_ptr，用於基本功能中parent和三個子程式的資料傳遞
3. Shm\_advmerge，用於進階功能中child和grandchild的資料傳遞
4. Shm\_advquick，用於進階功能中child和grandchild的資料傳遞

**< 灰底為 grandchild >**

S\_Main

Shm\_ptr🡺

S\_Main

Shm\_advmerge🡺

S\_Main

Shm\_advquick 🡺

主程式

Mchild[1]

Mchild[0]

Qchild[0]

Qchild[1]

## **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**

創建

Shm\_ptr🡺

S\_Main

Shm\_advmerge🡺

S\_Main

Shm\_advquick 🡺

主程式

S\_Main

1. 創建完 shared memory之後，會在 parent裡面 fork() 出 5個child
2. 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**

主程式

Shm\_ptr->Ready = 0

創建

停止

1. 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
2. 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
3. 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
4. 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**

1. 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 >**

## **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[2]**

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 )

## **程式如何編譯**

1. 有附上Makefile檔提供操作

OR

1. 在命令列輸入 : gcc -o 1043335\_01 1043335\_01.c –lrt
2. 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**

## **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