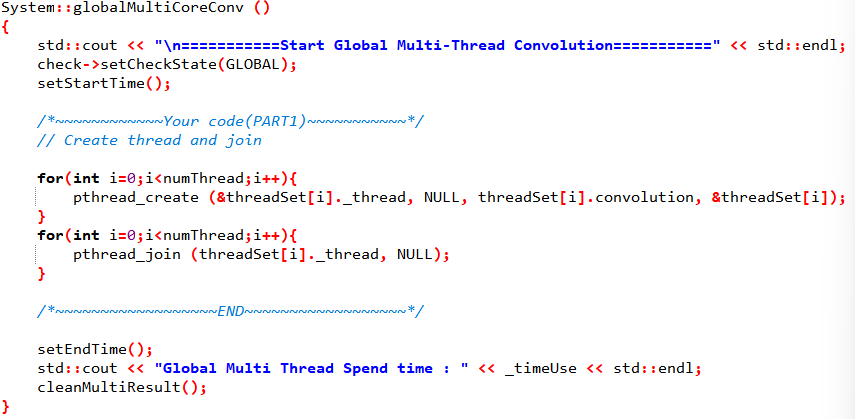
**嵌入式系統-PA1**

**學號：M11007326 姓名：黃鈞臨**

* **Part 1**

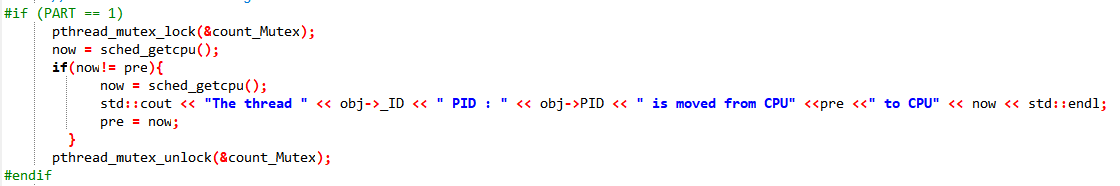
**[Global Scheduling 10%]**

* **Describe how to implement Global scheduling by using pthread. 5%**



用Pthread\_create建立新的執行緒，建立好的執行緒會以平行的方式執行，而最後再用Pthread\_join函數等待執行緒結束，而在設置Global scheduling時不必使用setCore設定cpu，因為setCore的預設值是-1，而在-1的情況下thread會配給到隨機的cpu上。

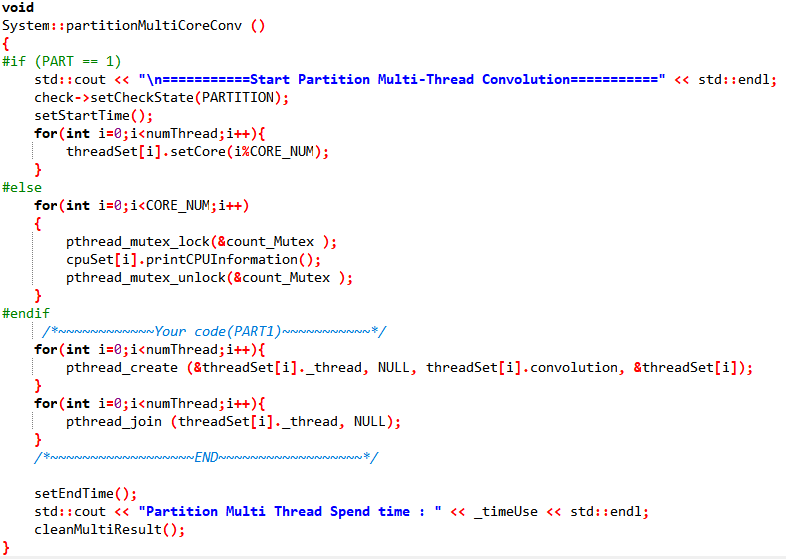
* **Describe how to observe task migration. 5%**

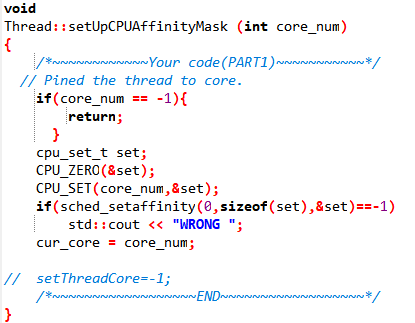
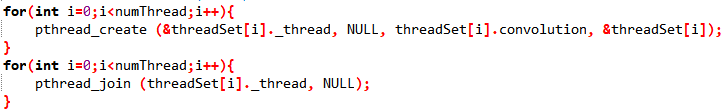


為了觀察CPU的變化，在進到convolution的for迴圈前，令兩個變數now跟pre並把當前的core丟進去，之後在for迴圈裡檢查cpu是否有變更，有的話就cout出來並更新pre。

**[Partition Scheduling. 5%]**

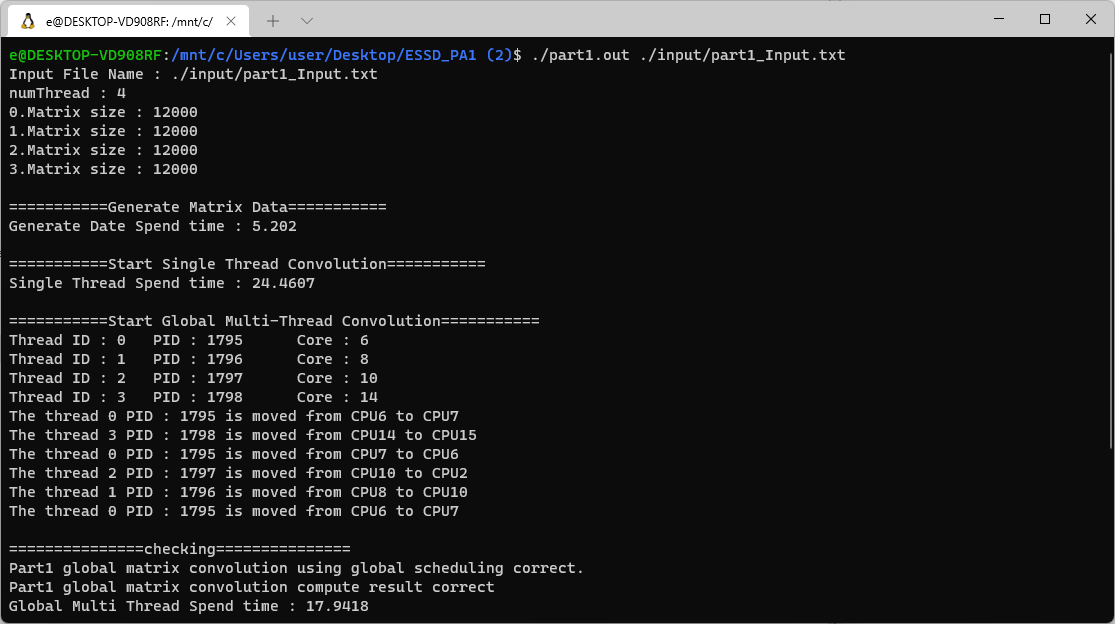
* **Describe how to implement partition scheduling by using pthread**



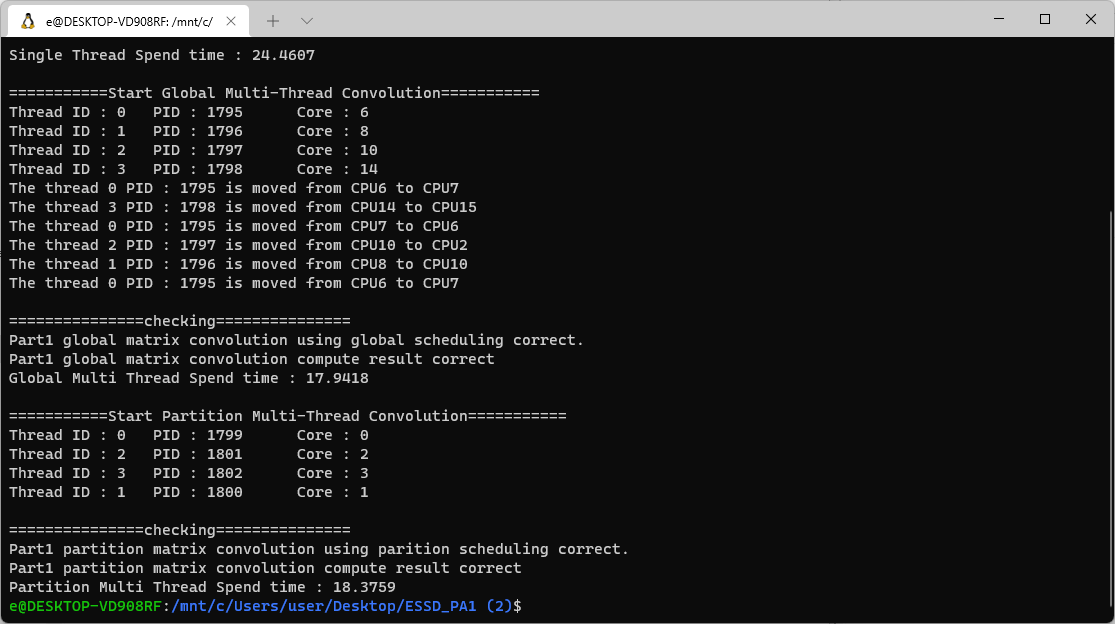


在partition時，跟global的差異在於，必須將thread分配給指定的cpu，因此這邊使用setCore將任務分配到指定的core上，之後在利用setUpCPUAffinityMask函式設置CPU。

**[Result. 10%]**

* **Show the scheduling states of tasks. (You have to show the screenshot result of using the input part1\_input.txt)**

▲Global Scheduling



▲Partition Scheduling

* **Part 2**

**[Partition method Implementation. 10%]**

* **Describe how to implement the three different partition methods (First-Fit, Best-Fit, Worst-Fit) in partition scheduling.**



First-Fit的排法要先將前面的cpu排滿後才排下一顆，因此這邊用一個while判斷是否排滿，排滿的話就排下一顆，而排到超過指定的CPU數量後，就判斷這個任務無法排程，並把core設成-1。



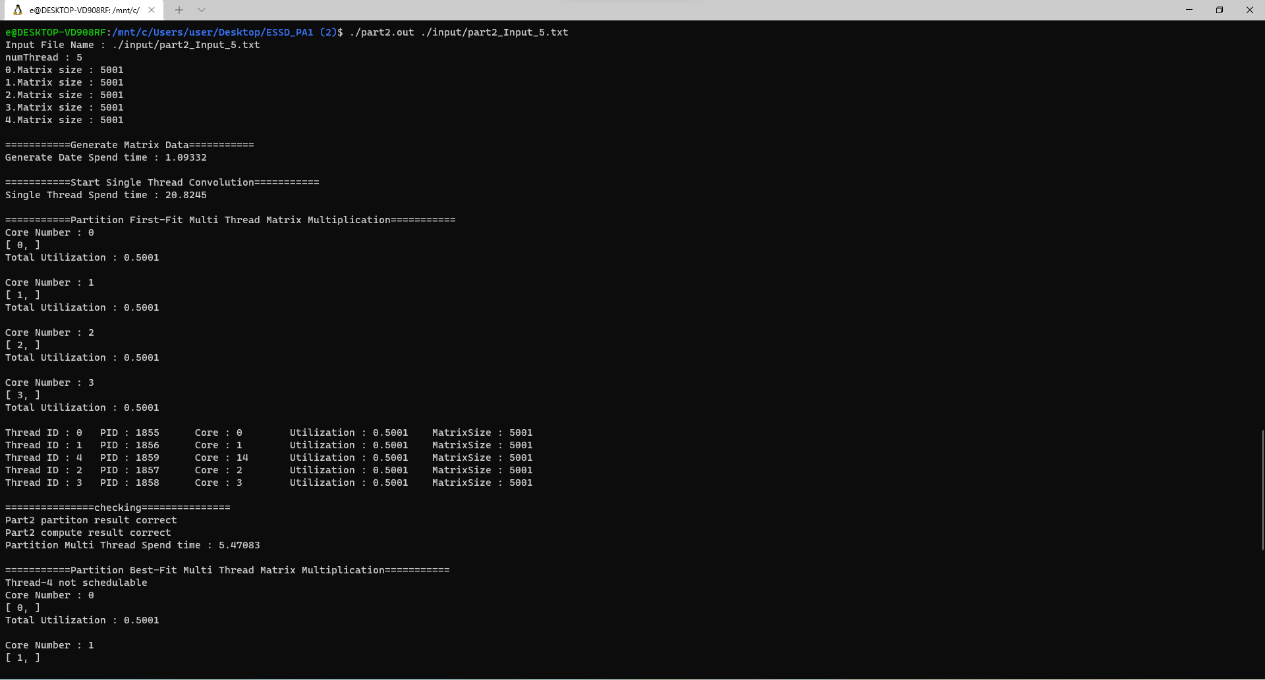
Best-Fit的排法要先把當前利用率最高的先排滿，因此必須在排每個任務時都去比較每個cpu的使用率，上圖第216行的迴圈就是在比較cpu的使用率，而當cpu的超過指定的數量時，判斷任務無法排成，並將core設成-1。



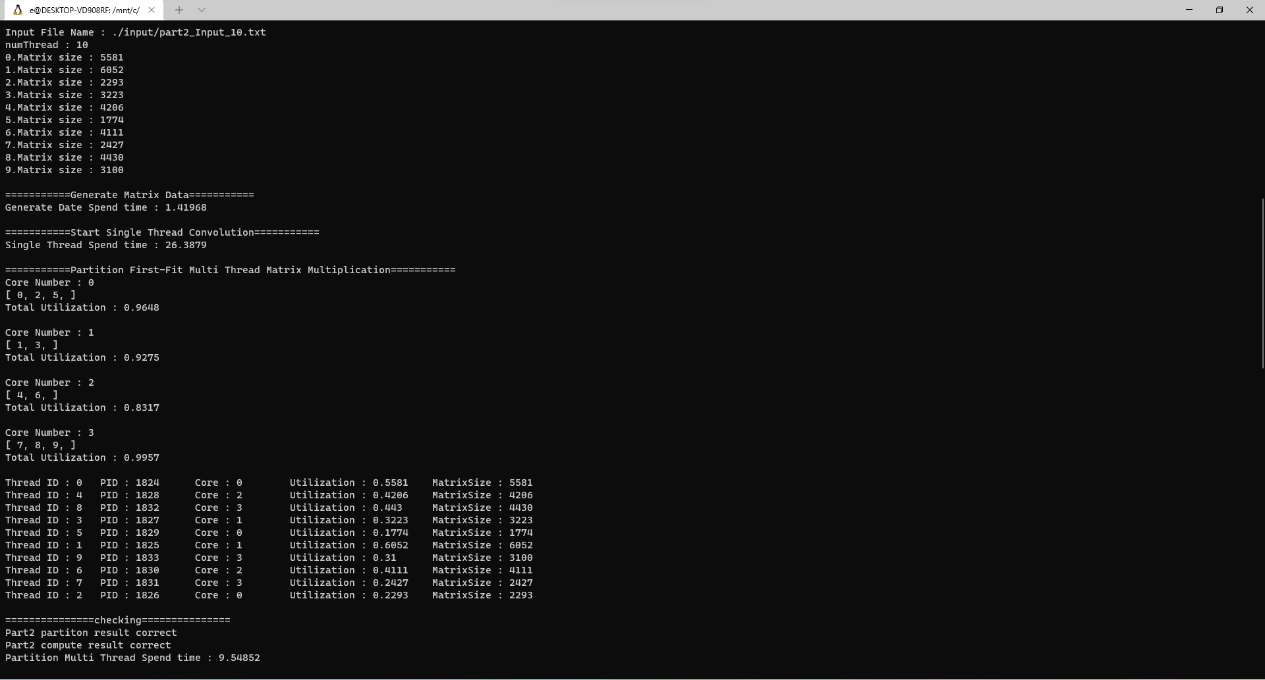
Worst Fit的排法是先選擇利用率最小的cpu排程，與Best Fit 相反，在程式上也差不多，只是將原本找出最大的cpu改成找出最小的cpu，其他就都一樣。

**[Result. 30%]**

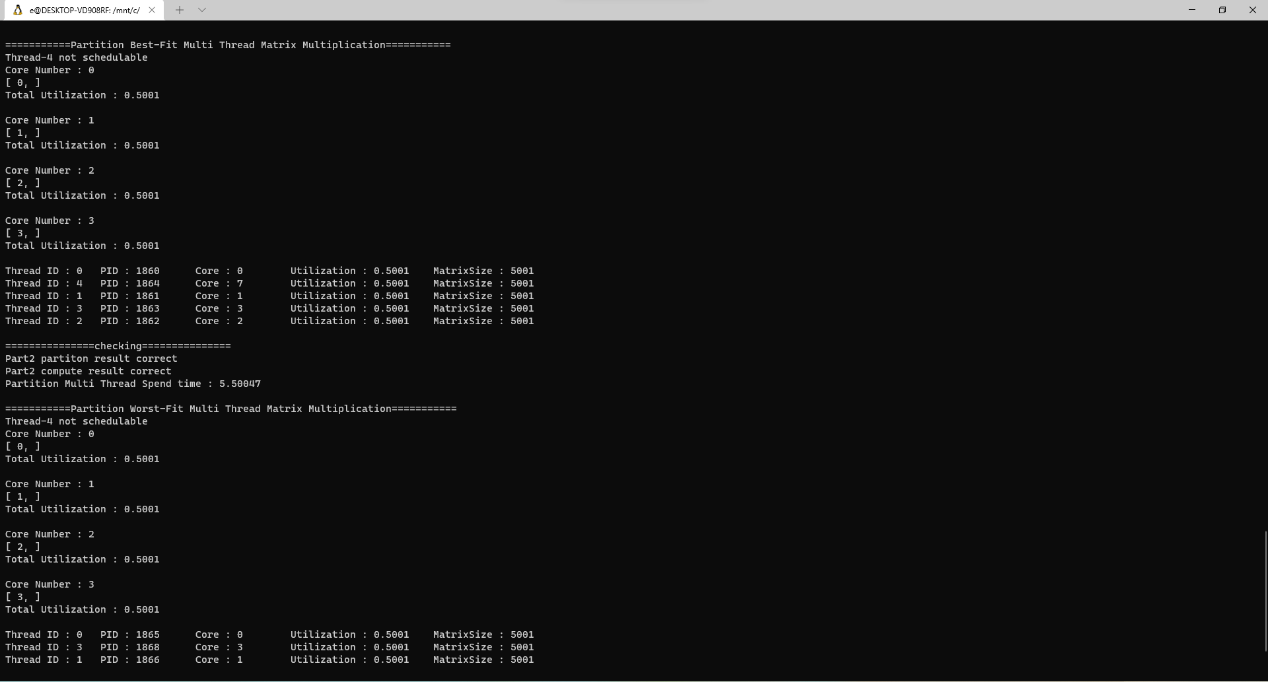
* **Show the scheduling states of tasks. (You have to show the screenshot result of using input part2\_input\_5.txt and part2\_input\_10.txt)**



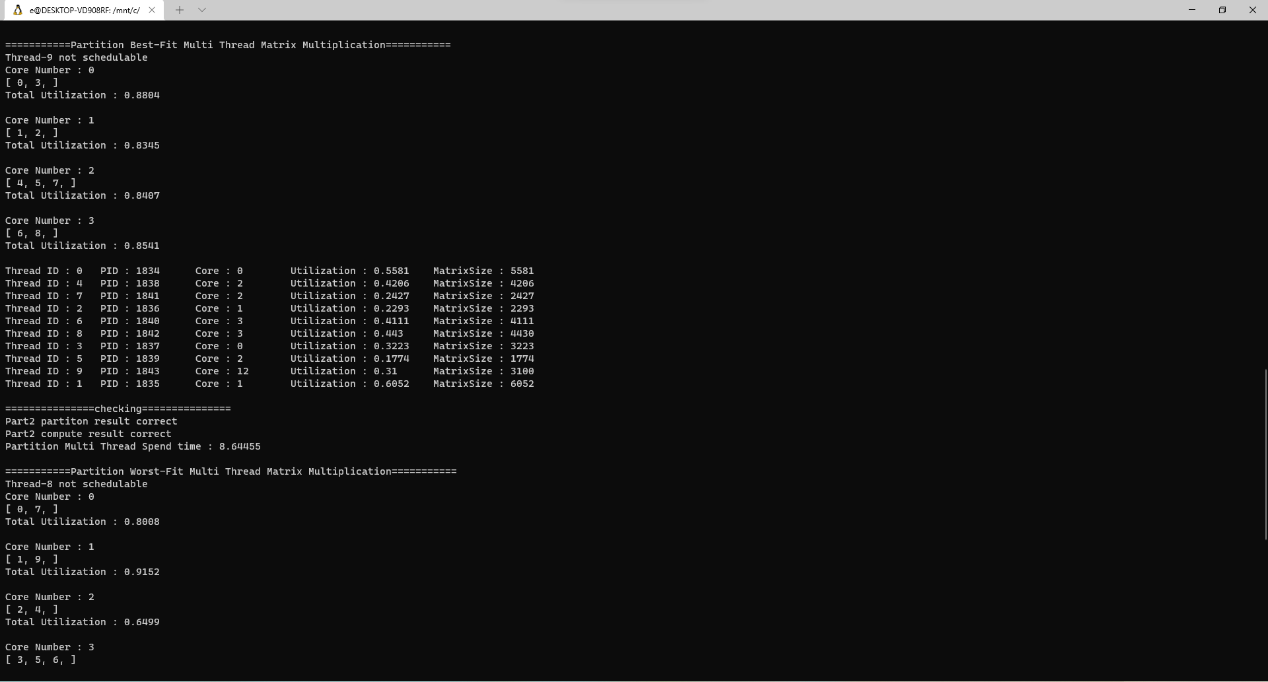
▲First-Fit with input\_5.txt



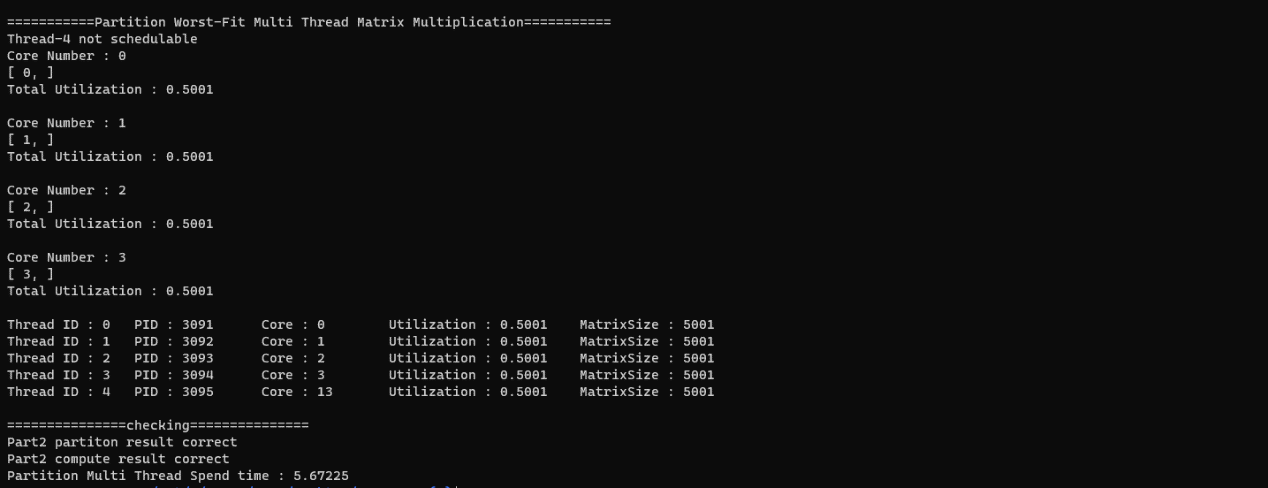
▲First-Fit with input\_10.txt



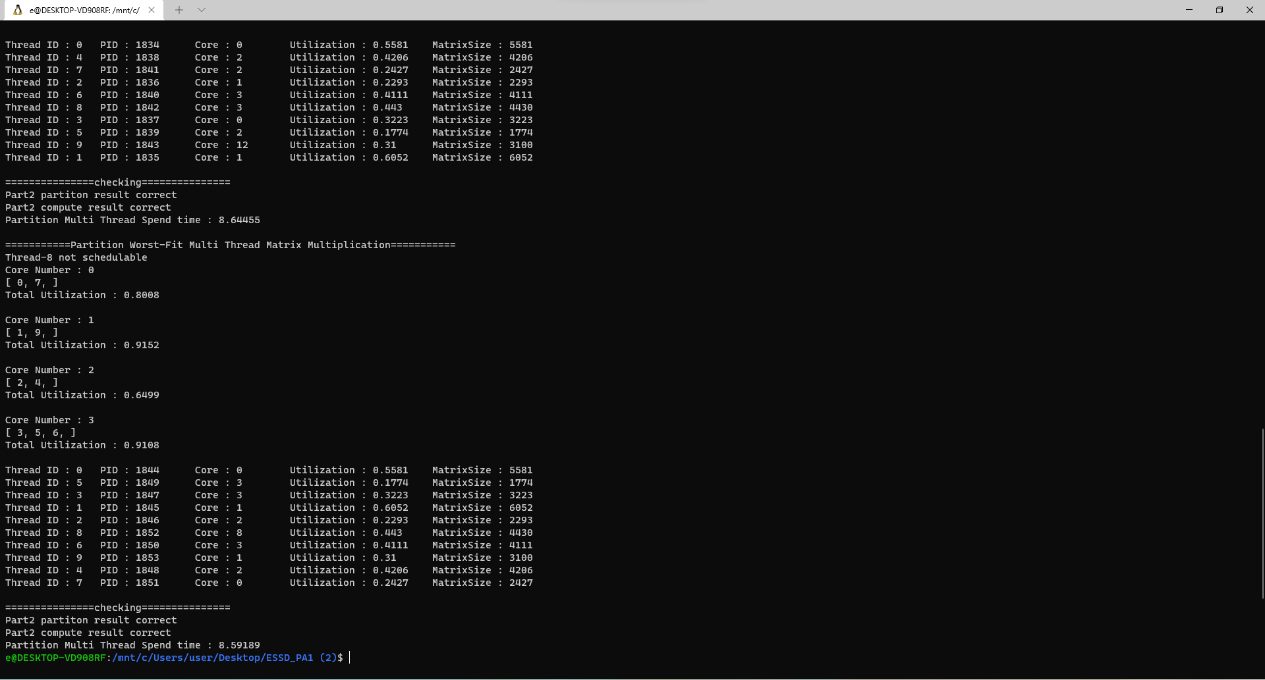
▲Best-Fit with input\_5.txt



▲Best-Fit with input\_10.txt



▲Worst-Fit with input\_5.txt

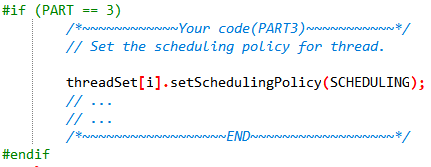


▲Worst-Fit with input\_10.txt

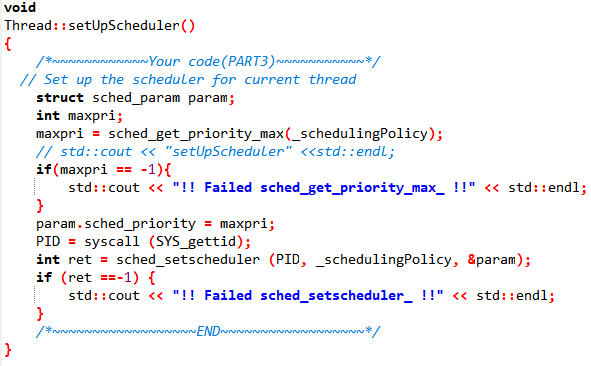
* **Part 3**

**[Scheduler Implementation. 10%]**

* **Describe how to implement the scheduler setting in partition scheduling. (FIFO with FF, RR with FF)**



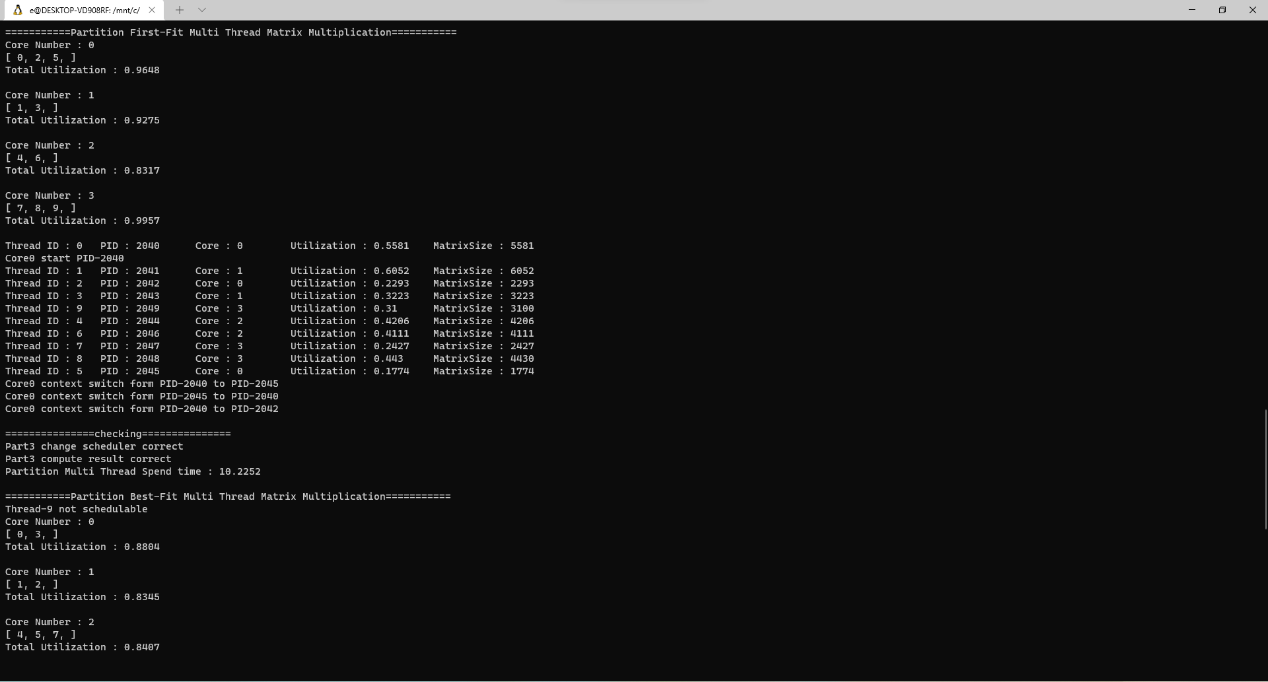
首先在system.cpp中根據讀進來的SCHEDULING設定FIFO或是Round-Robin



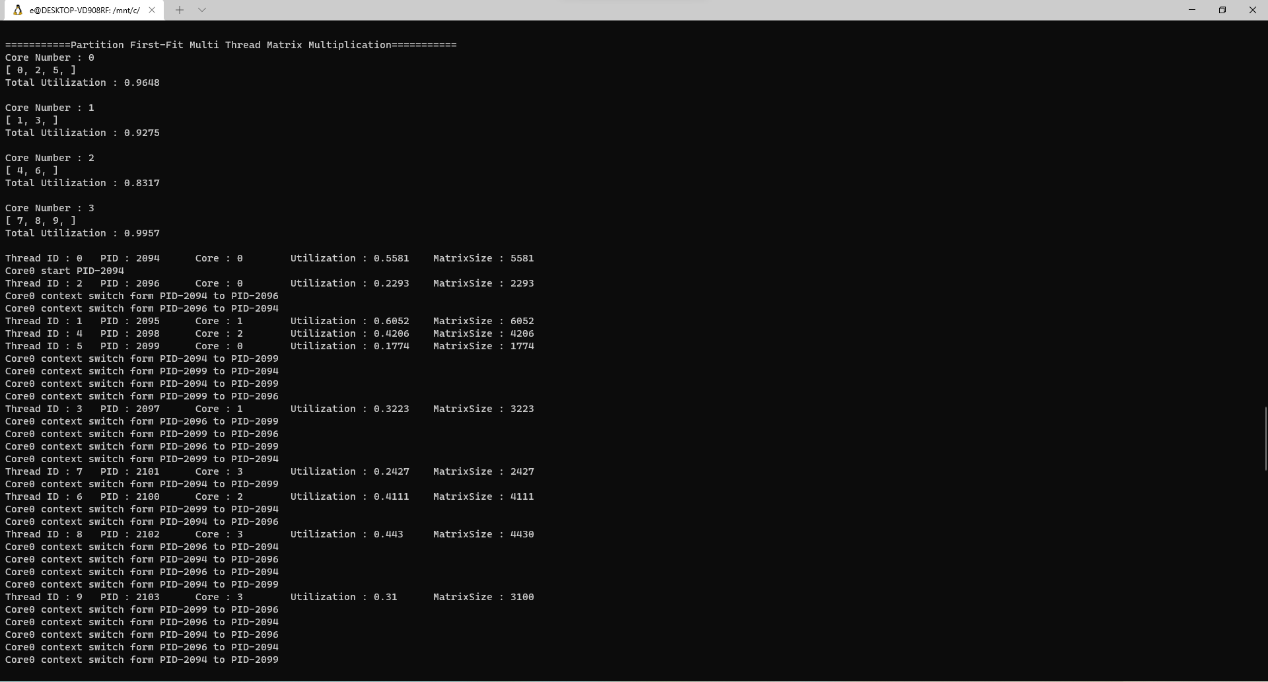
接著在setUpScheduler裡的sched\_get\_priority\_max()拿到該排程的最大優先權，再透過sched\_setscheduler (PID, \_schedulingPolicy, &param);設置排程就完成了。

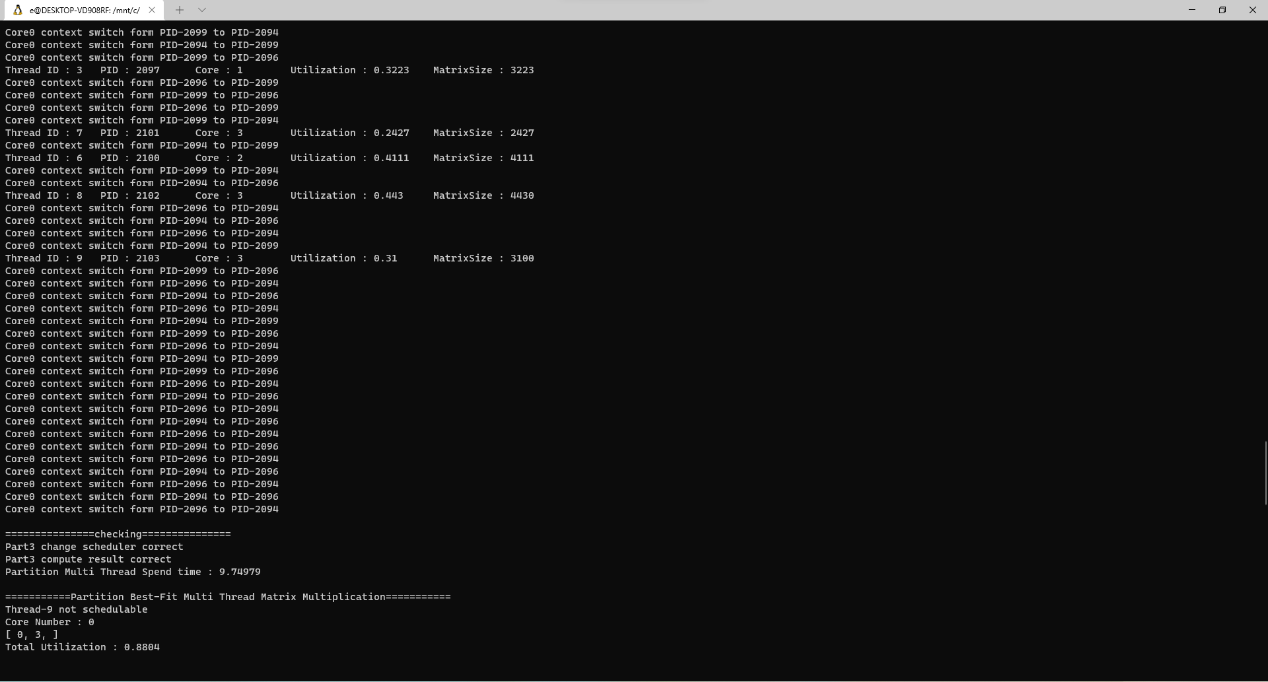
[Result. 10%]

* **Show the process execution states of tasks. (You have to show the screen shot result of using input part3\_input.txt)**



▲FIFO





▲Round-Robin

* **Discussion**
* **Analyze and compare the response time of the program, with single thread and multi-thread using in part 1 and part 2. (Including Single, Global, First-Fit, Best-Fit, Worst-Fit) 5%**

首先使用多核的執行時間比起單核快上不少，而在GLOAB時我發現我的電腦在全部的矩陣大小都是一萬時print不出core切換的資訊，可能是電腦的核心比較好的原因。

而First-Fit、Best-Fit和Worst-Fit三個排法中，在輸入為10的時候，最快的是Worst-Fitst，不過這是因為他有一個TASK是給額外的core做，因此比較快也是蠻合理的。

* **Analyze and compare the characteristic of the three different partition methods (First-Fit, Best-Fit, Worst-Fit) 5%**

在10個thread中，First-Fit是唯一能夠全部排進去的排法，最滿的利用率是0.9957，最少的則是0.8317，接下來Best-Fit的排法沒辦法將最後一個任務排程，最大的利用率是0.8804，最少的則是0.8345，最後的Worst-Fit則是無法將倒數第二個任務排程，最大的利用率是0.9152，最少的則是0.6499，First-Fit的排法跟進來的任務順序有很大的關係，有可能是剛好塞的下，Best-Fit的排法是先將最大使用率的core塞滿，理論上感覺可以省下最多的利用率，但從結果來看分配還蠻平均的，我認為主要是任務的利用率都不算小，因此常發生塞不下的情況，最後，Worst-Fit的排法是先排進利用率最小的core，理論上分配能夠最平均，但結果看來差挺多的，主要是從第七個任務(Thread 6)開始放的任務量差蠻多的，只看前六個任務的話還蠻平均的。

* **Analyze and compare the response time of the program, with two different schedulers. (FIFO with FF, RR with FF) 5%**

FIFO的context switch次數明顯比RR少非常多，這是因為FIFO在執行完當前任務前不會切換給下一個任務，而RR則是每個任務會輪流執行，比較執行時間的話，我的結果是RR比FIFO快0.5秒，不過這個差距有可能是受到電腦當時的背景程序影響，否則我認為context switch比較少的FIFO應該會快一點才對。