嵌入式系統軟體設計

Embedded System Software Design

PA1

指導教授: 陳雅淑 教授

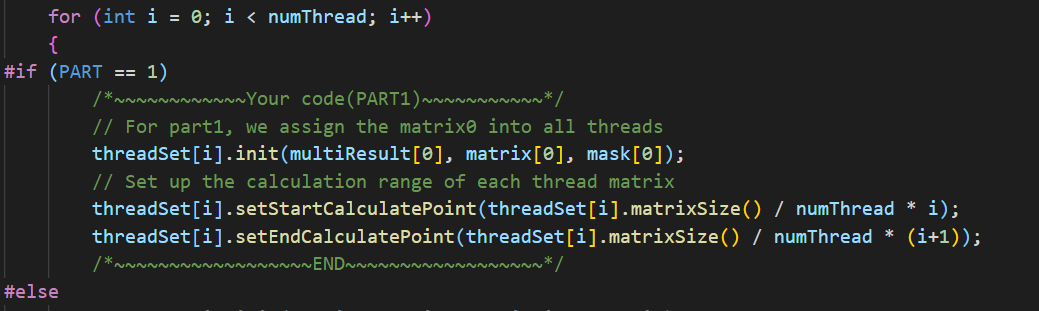
學生: M11007308 吳柏翰

**Part 1**

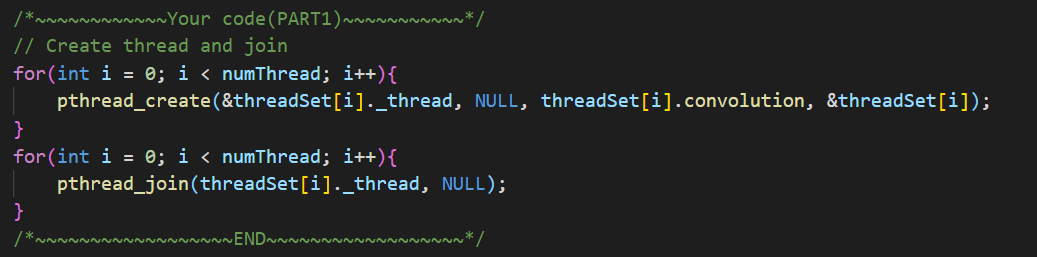
**[Global Scheduling. 10%]**

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

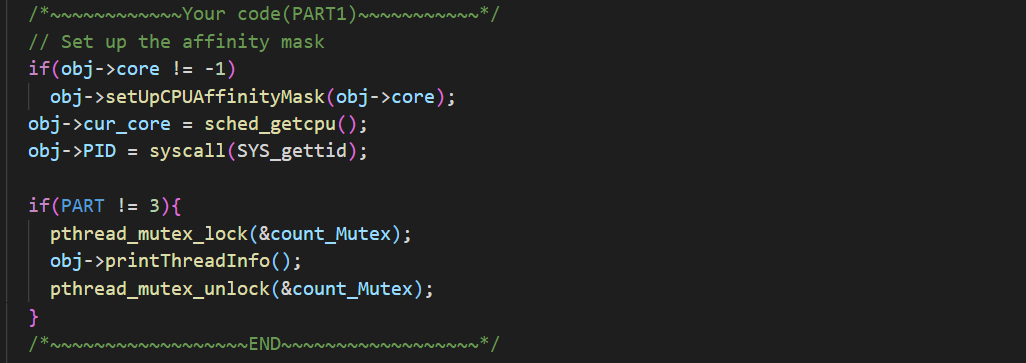
首先要定義每個執行緒(thread)計算的範圍，以本例而言，要計算的矩陣大小為10000，平均分派給4個執行緒，所以執行緒各被分配到2500。



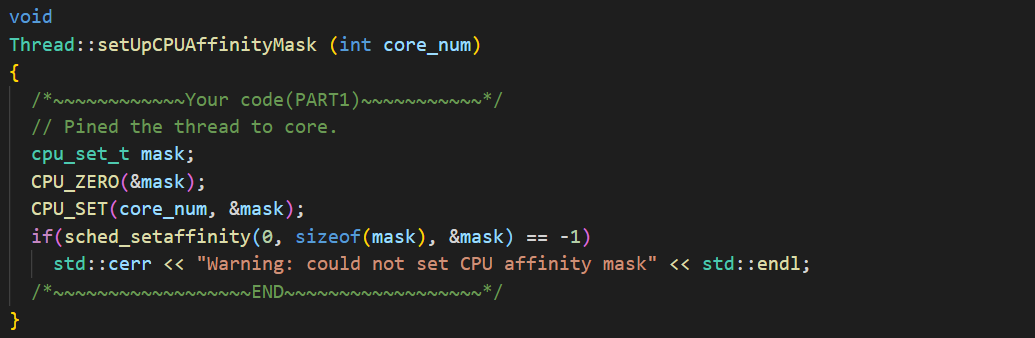
pthread的pthread\_create函數可以用來建立新的執行緒，並指定子執行緒要執行的函數，子執行緒在建立之後，就會以平行的方式執行，在子執行緒的執行期間，主執行緒還是可以正常執行自己的工作，最後主執行緒再以 pthread\_join 函數等待子執行緒執行結束，處理後續收尾的動作。



若利用多執行緒進行運算，就需要設定Affinity mask，決定哪個CPU可以執行程式，讓系統不要自動排程。如果指定好核心(core)的話就執行 setupCPU-AffinityMask，並更新目前的執行緒是在哪個core中及thread ID，再顯示執行緒資訊。

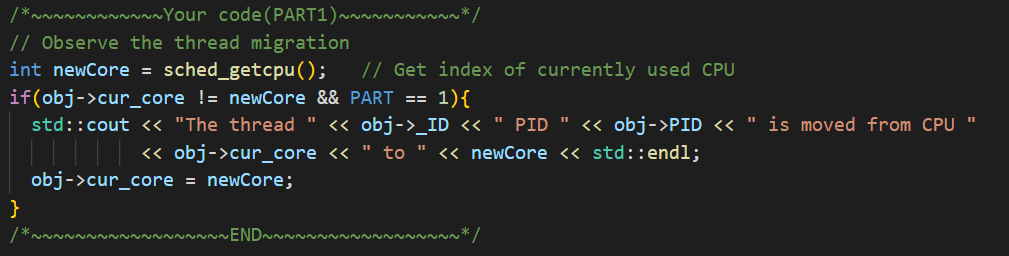


以下是設定Affinity mask的副程式，將thread固定在指定的Core上。



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

當執行緒目前被某個core執行，若和上一次紀錄的core不同時，就表示發生Task migration，就顯示該執行緒是從哪個core移動到其他哪個core。



**[Partition Scheduling. 5%]**

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

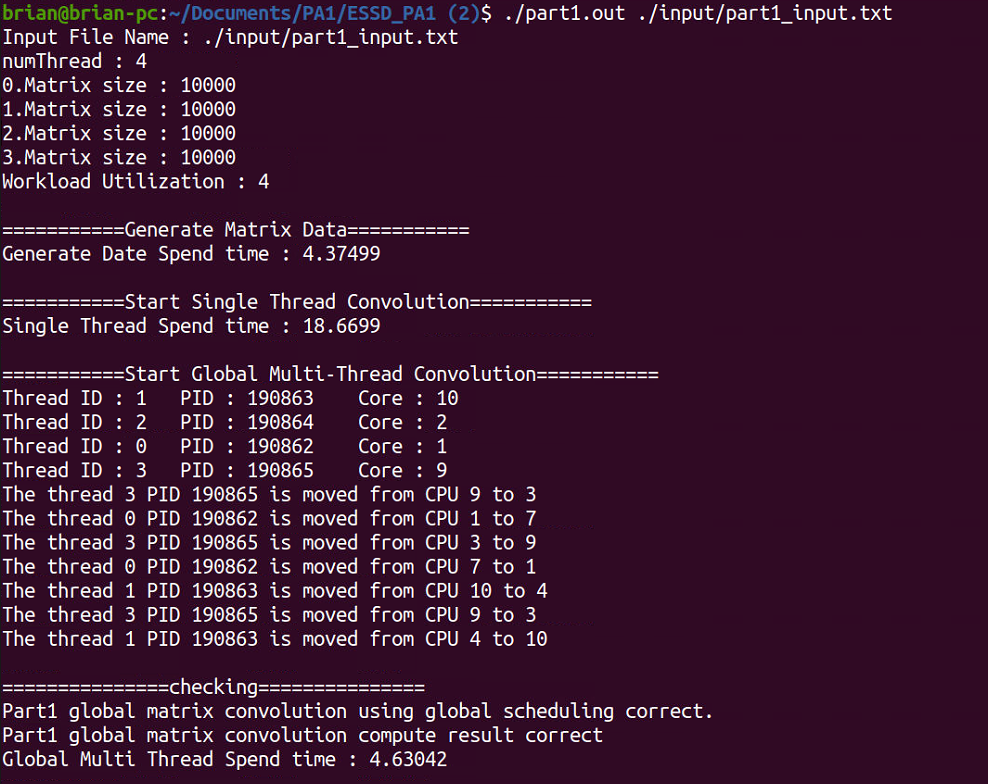
Partition scheduling與Global scheduling不同的地方是: Partition scheduling需要預先將thread指定給哪個core，這裡就將第i個thread指定給第i個core。



**[Result. 10%]**

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

(1)Single thread與Global multi-thread scheduling的結果



(2)Partition multi-thread scheduling的結果

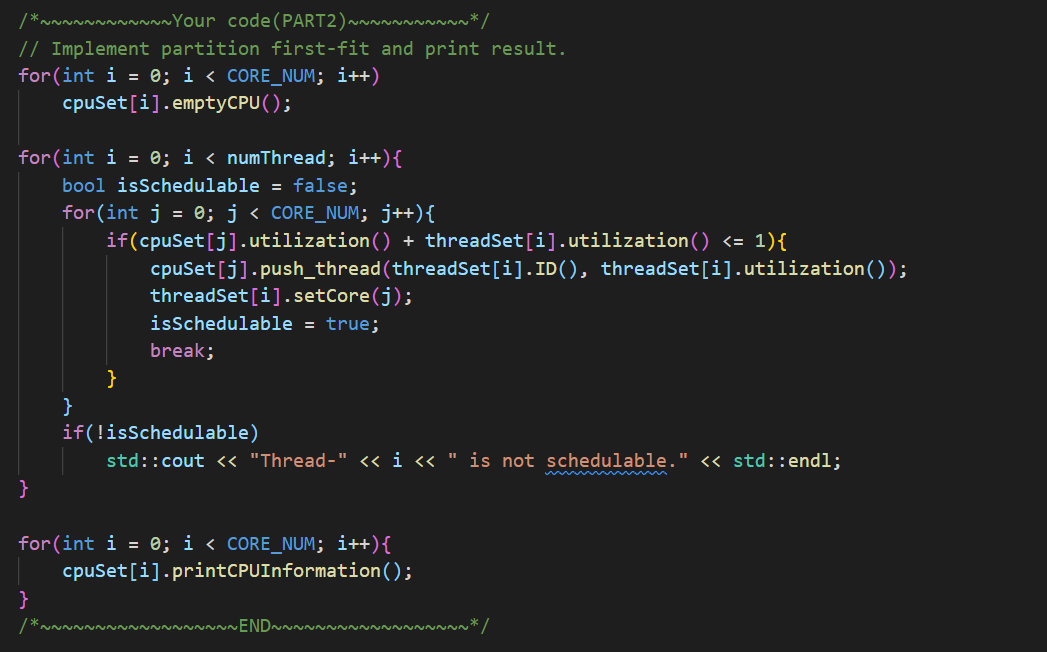


**Part 2**

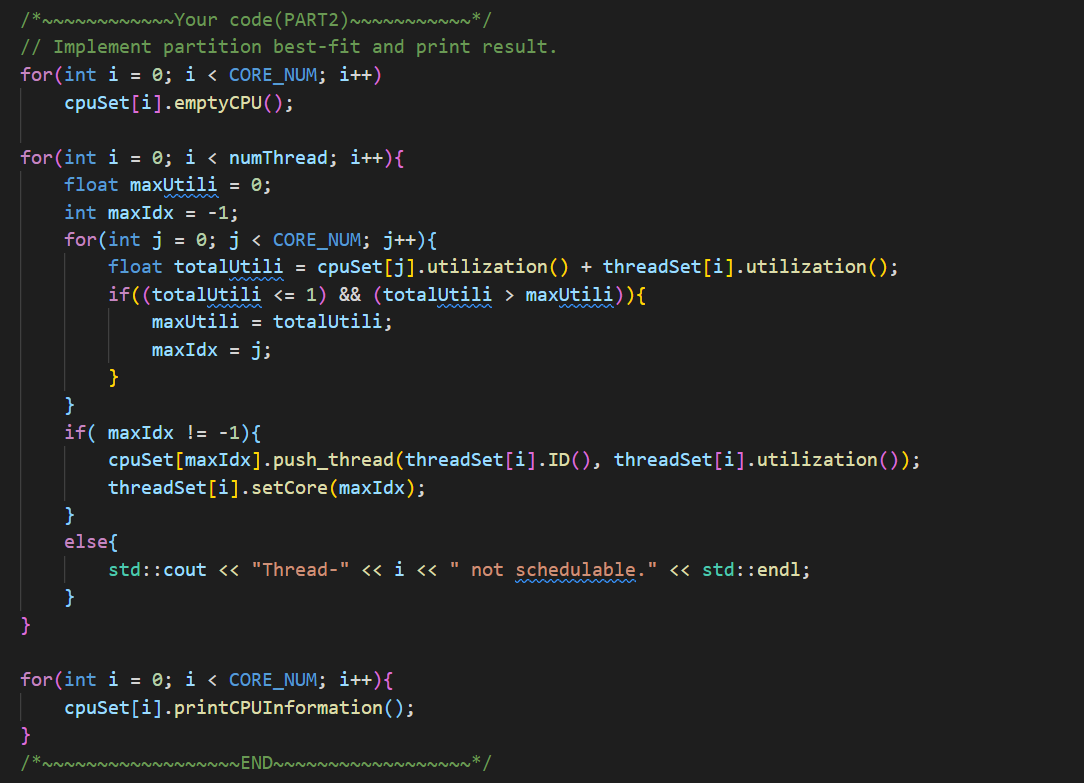
**[Partition method Implementation. 10%]**

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

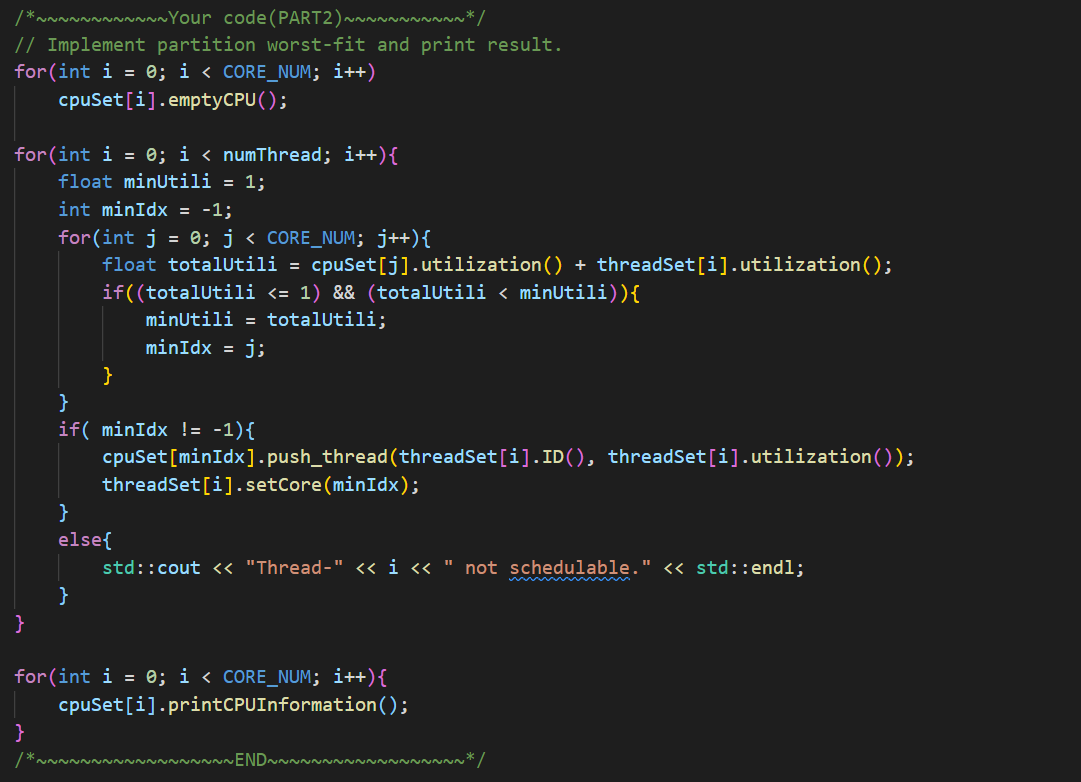
First-fit:優先將執行緒排入較低index的核心(core)中，如果第一個核心的使用量已被塞滿或是排入時使用量會大於1，則接續排入下一個核心，最後顯示排程結果，若有執行緒未能排入，則顯示該執行緒不能被排程。



Best-fit:優先將執行緒排入使用率最高的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於1，則接續排入使用量次高的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最高的核心並排入，最後顯示排程結果，若有執行緒未能排入，則顯示該執行緒不能被排程。



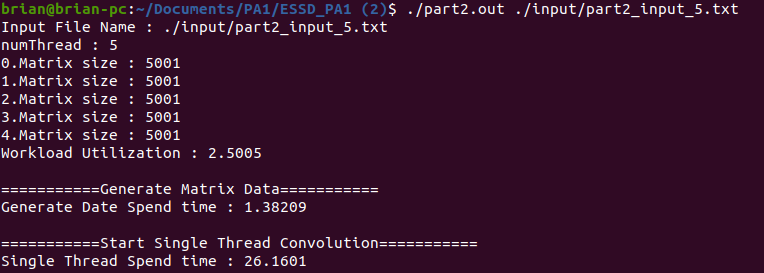
Worst-fit:優先將執行緒排入使用率最低的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於1，則接續排入使用量次低的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最低的核心並排入，最後顯示排程結果，若有執行緒未能排入，則顯示該執行緒不能被排程。



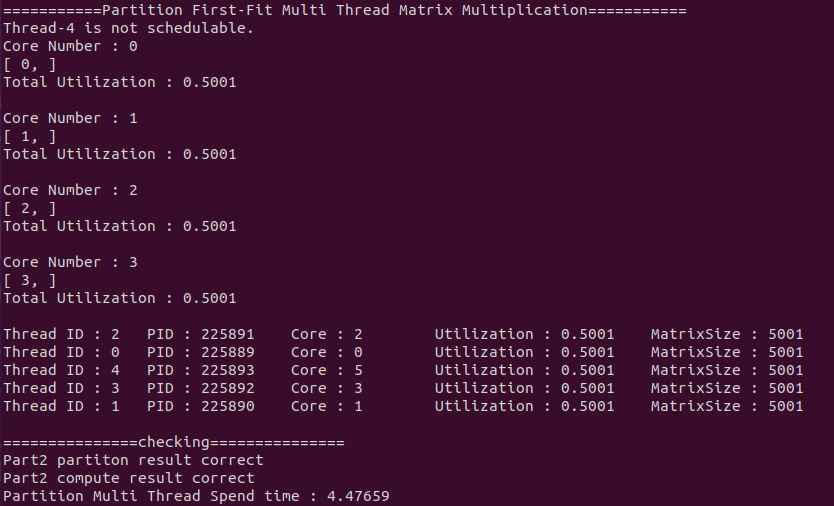
**[Result. 30%]**

* **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)**

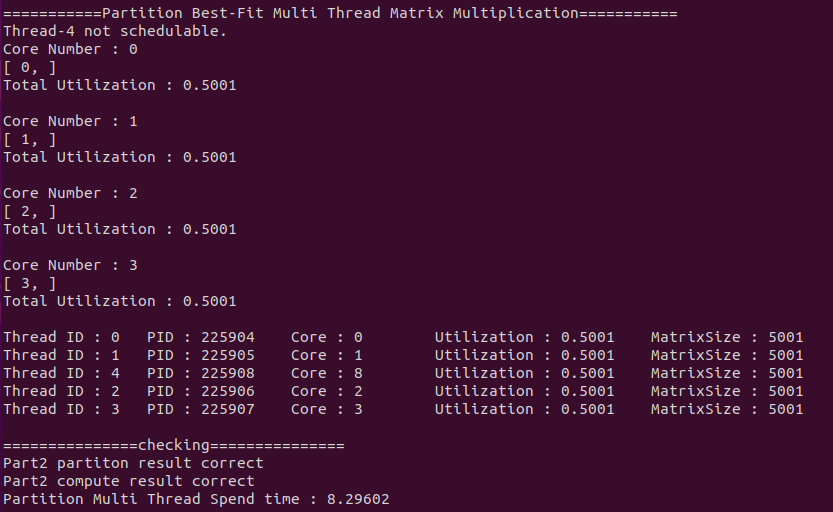
1. part2\_input\_5.txt的結果



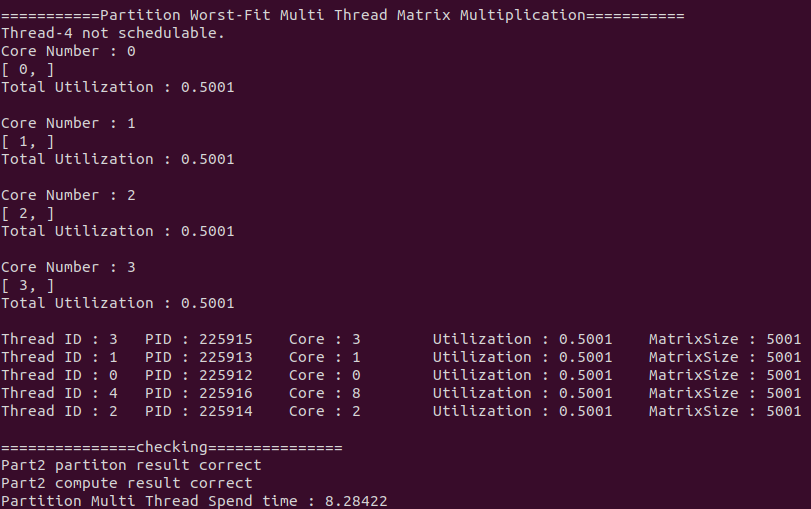
First-fit排程結果



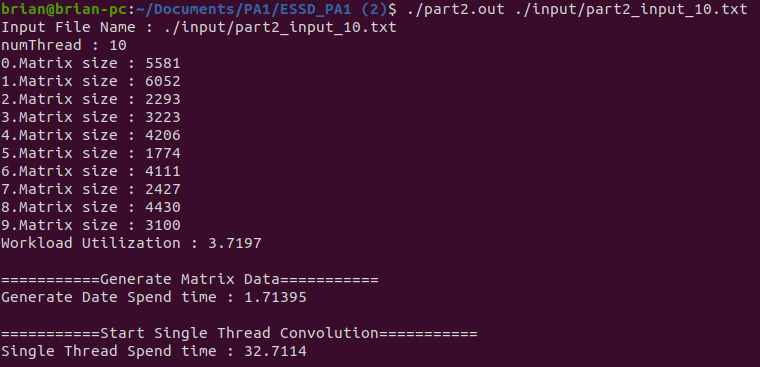
Best-fit排程結果



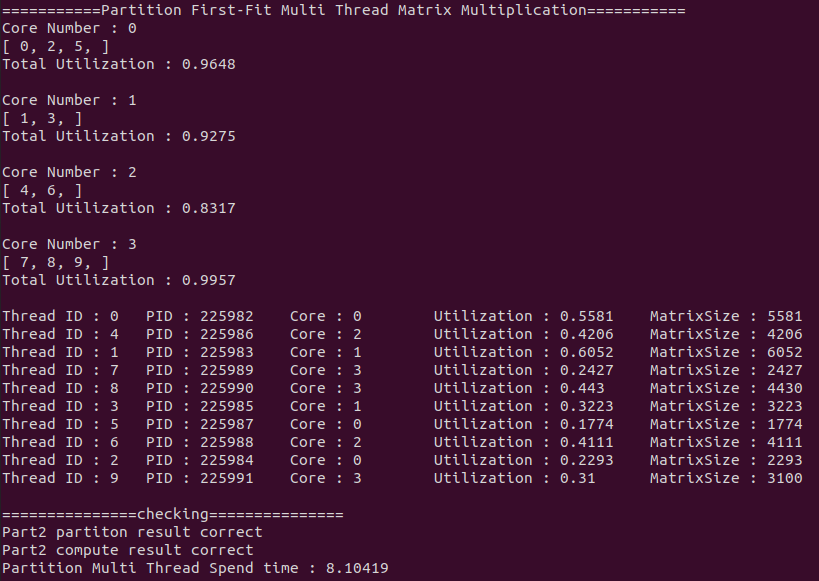
Worst-fit排程結果



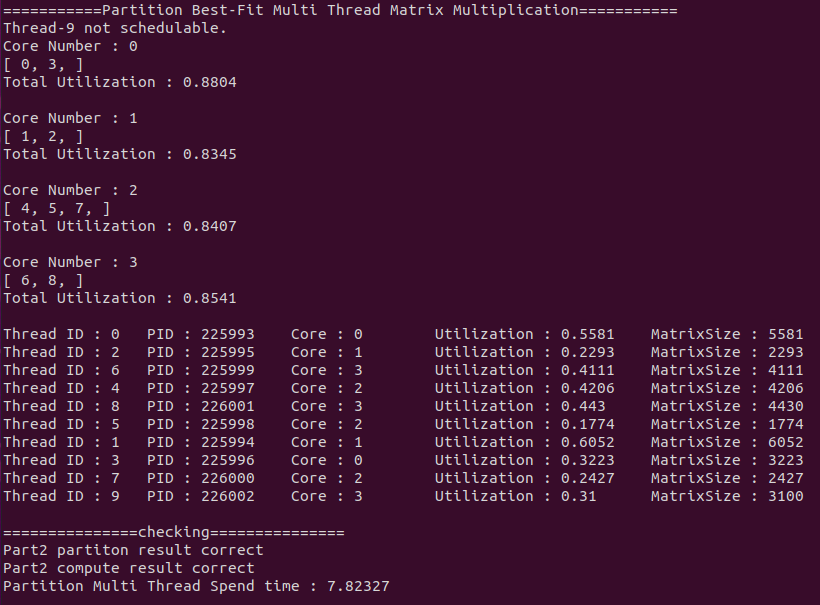
1. part2\_input\_10.txt的結果



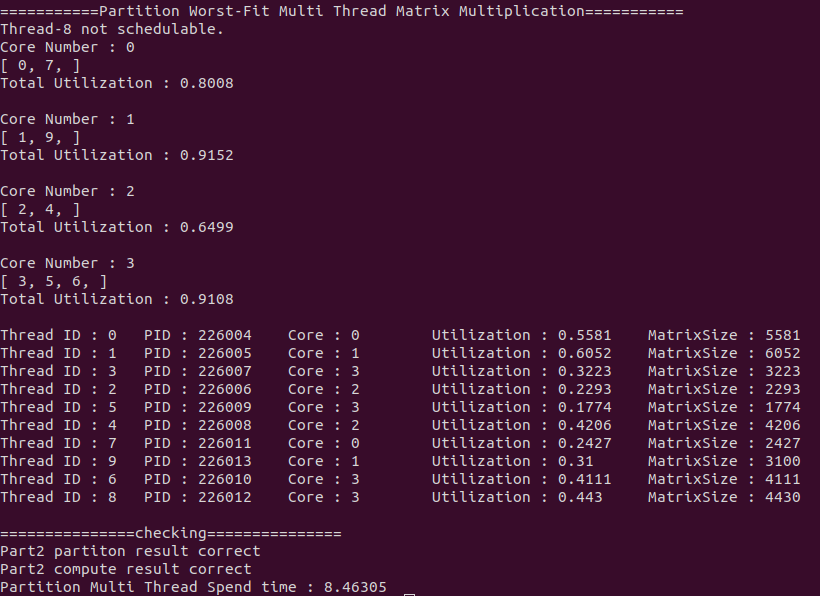
First-fit排程結果



Best-fit排程結果



Worst-fit排程結果

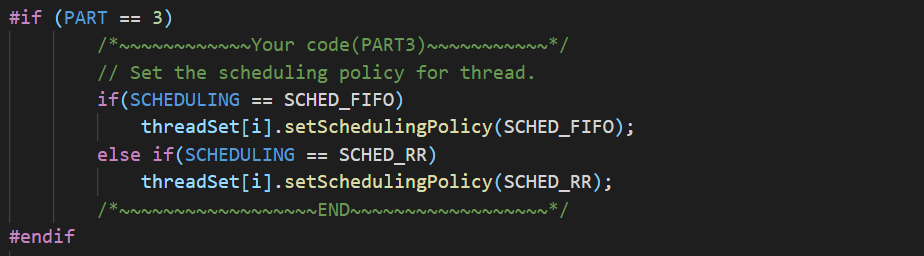


**Part 3**

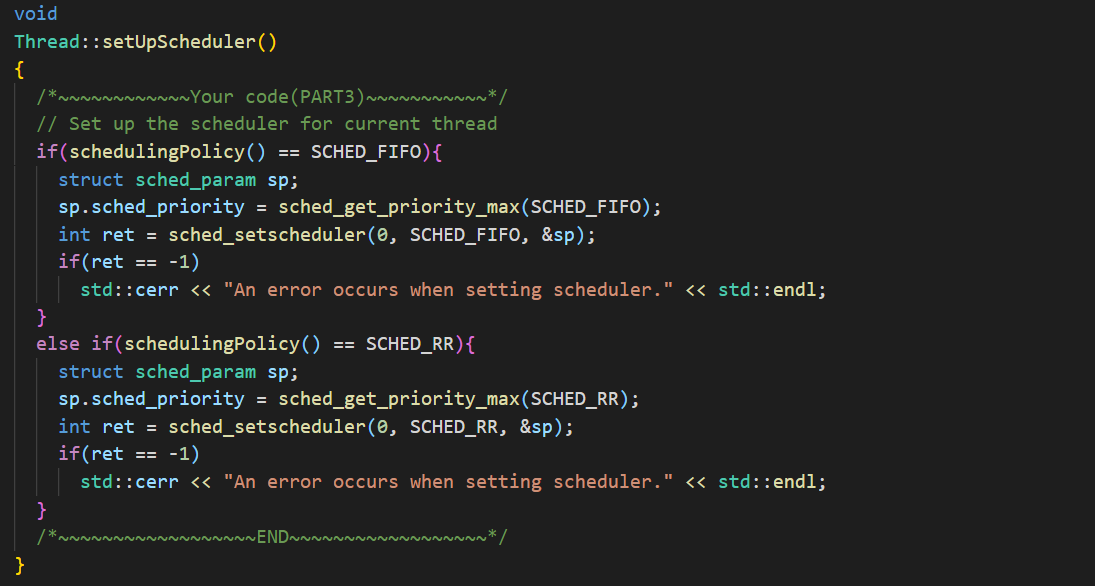
**[Scheduler Implementation. 10%]**

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

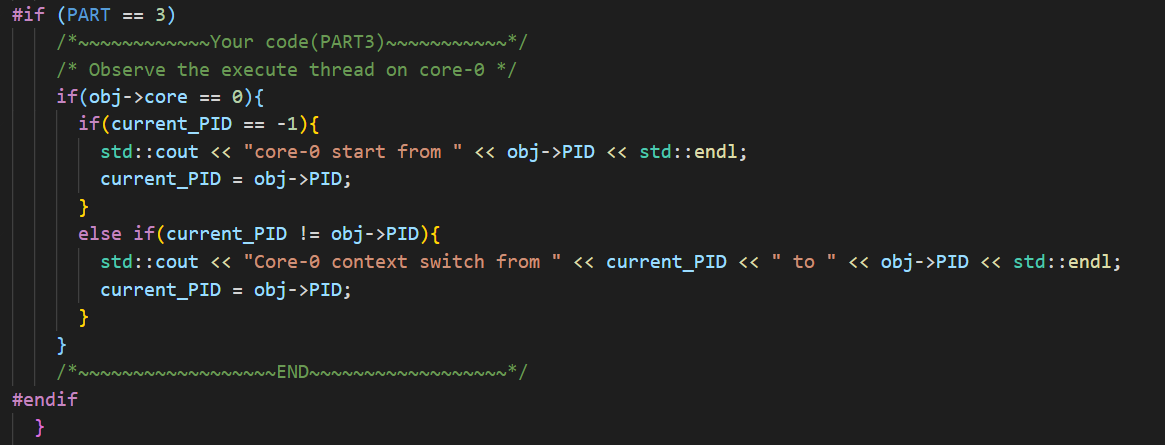
一開始初始化thread後，設定thread的排程方式是FIFO或Round-Robin。



利用Linux支援的排程器，像是FIFO和RR，再使用sched\_setscheduler函數設定排程規則。



顯示core-0的資訊，以及發生context switch的情況。

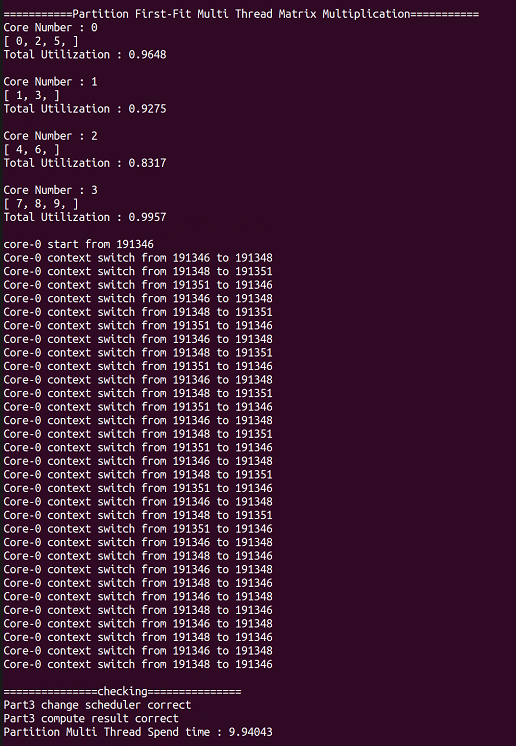


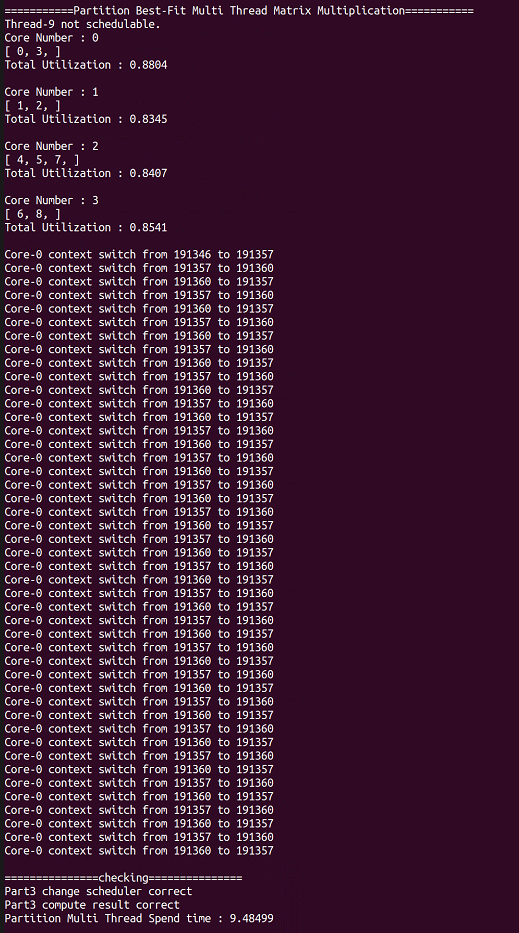
**[Result. 10%]**

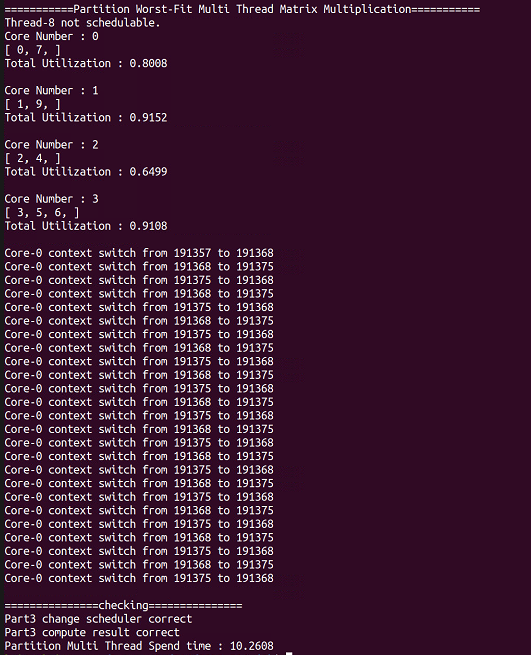
* **Show the process execution states of tasks. (You have to show the screen-shot result of using input part3\_input.txt)**

(1)part3\_input.txt (Round-Robin)的結果





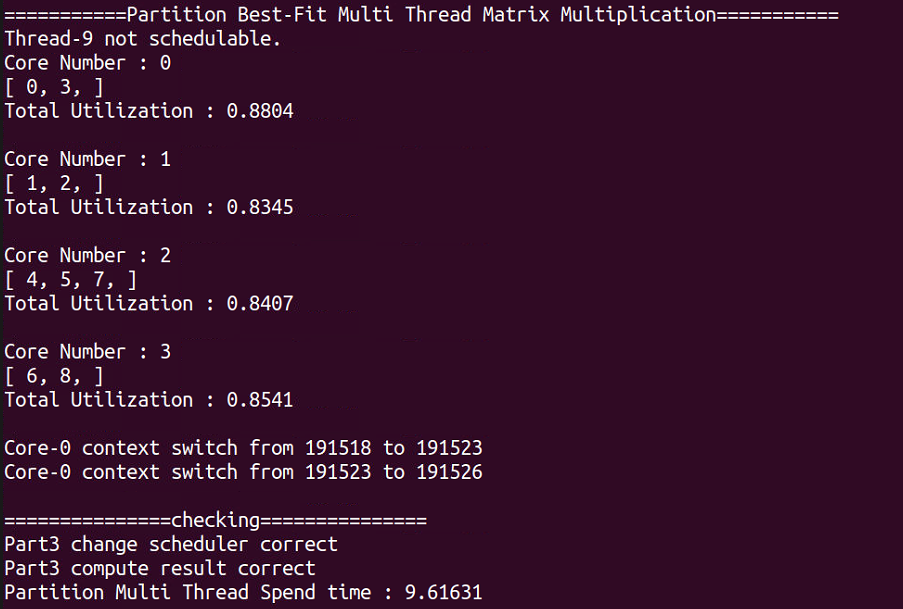


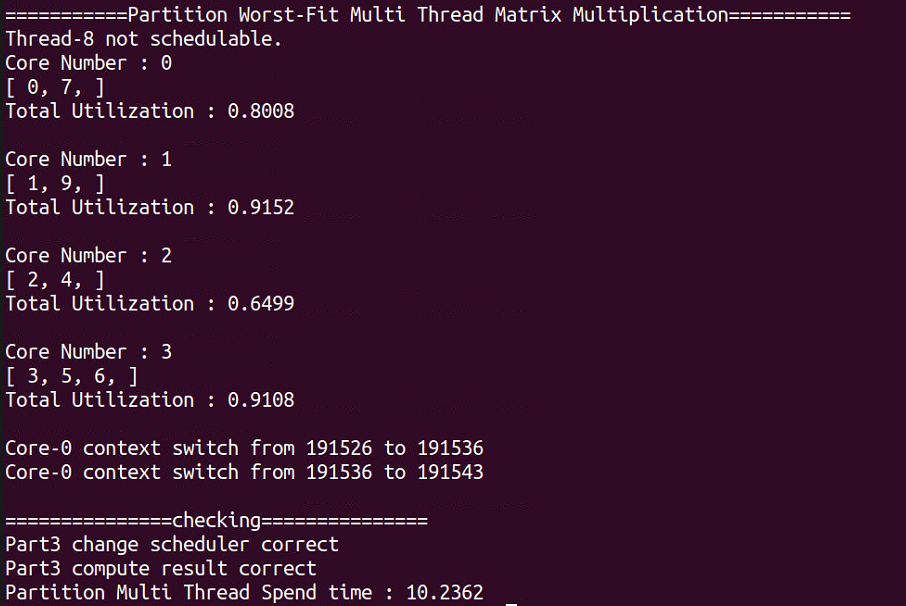


(1)part3\_input.txt (FIFO)的結果









**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%**

1. part1\_input.txt的測試結果

|  |  |  |  |
| --- | --- | --- | --- |
|  | Single | Global | Partition |
| Time (sec) | 18.6699 | 4.6304 | 4.4654 |

(2) part2\_input\_5.txt的測試結果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Single | First-fit | Best-fit | Worst-fit |
| Time (sec) | 26.1601 | 4.4766 | 8.2960 | 8.2842 |

(3) part2\_input\_10.txt的測試結果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Single | First-fit | Best-fit | Worst-fit |
| Time (sec) | 32.7114 | 8.1042 | 7.8233 | 8.4631 |

從Part1的結果可得知在速度上Partition> Global> Single，Multi-threading的運行速度遠快於Single-threading，因為多個執行緒同時運算一定比單個執行緒來的更有效率，另外在運算時間上Partition scheduling比Global scheduling快一些，因為Global scheduling會將所有task先存放在queue，再依序分配給CPU執行;而Partition scheduling會直接將task分配給CPU執行。

從Part2的第一項結果可看出First-fit的速度是最快，但第二項結果顯示Best-fit是最快，理論上First-fit應該是最快，因為在演算法上較為單純，每次在排入task時都是從較小index的CPU開始排入，Best-fit和Worst-fit就需要比較各個CPU的利用率再排入task，與First-fit相比會慢一些。但是在經過反覆測試後，我發現這三種排法所花費的時間都不固定，有時是First-fit最快，有時是Best-fit或Worst-fit，推測應該是有某個thread沒辦法排入，所以少了一點排入所花費的時間。

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

First-fit: 優先將執行緒排入較低index的核心(core)中，如果第一個核心的使用量已被塞滿或是排入時使用量會大於1，則接續排入下一個核心。優點是執行速度最快，且演算法簡單容易實現。

Best-fit:優先將執行緒排入使用率最高的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於1，則接續排入使用量次高的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最高的核心並排入。特點是可集中使用某些GPU，讓剩餘沒有被排入工作的CPU降低工作量。

Worst-fit:優先將執行緒排入使用率最低的核心(core)中，如果該核心的使用量已被塞滿或是排入時使用量會大於1，則接續排入使用量次低的核心，若還是不能排入，則接續和其他核心比較使用量，選出之中使用量最低的核心並排入。特點是平均使用每個CPU，其利用率最平均，對CPU而言功耗最少。

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

|  |  |  |
| --- | --- | --- |
|  | RR with FF | FIFO with FF |
| Time (sec) | 9.9404 | 9.8678 |

由測試結果可發現FIFO比RR快一些，RR需要一直做context switch，要花費較長的時間處理context switch;而FIFO是要等待上一個thread執行完才會執行下一個thread，若前一個thread長時間佔住CPU，就會導致下一個thread遲遲無法執行，可能花更多時間在等待上。