嵌入式系統軟件設計項目1

問題定義：

通過並行執行程序中的某些區域，我們可以減少執行時間。 在本項目中，要求您觀察Linux中由POSIX線程執行的單線程和多線程程序的性能，並觀察由全局和分區管理的此類程序的響應時間（優先，最佳，最差）。 -Fit）在Linux系統中具有不同的調度程序（FIFO，循環）。

實驗環境：

✓PC：至少4核

✓RAM：至少4GB

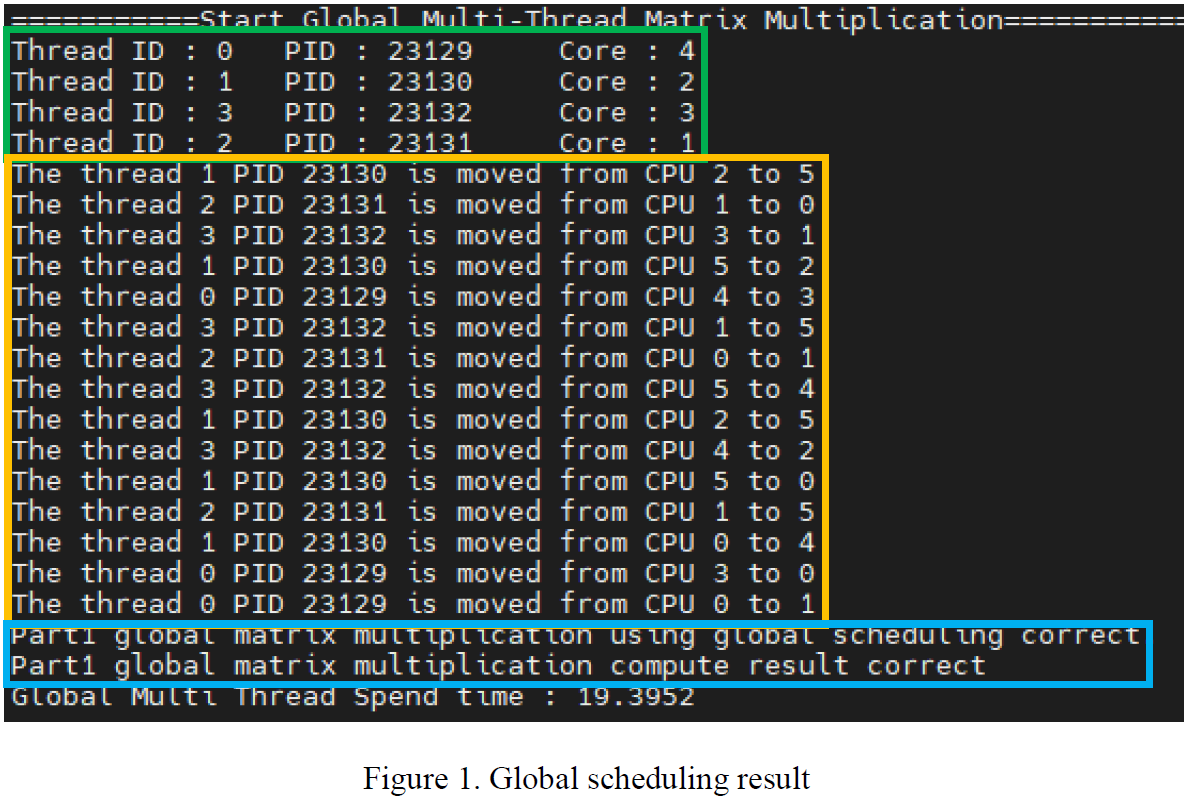
✓操作系統：Ubuntu 16.04或更高版本

✓編譯器：g ++ 7.5.0

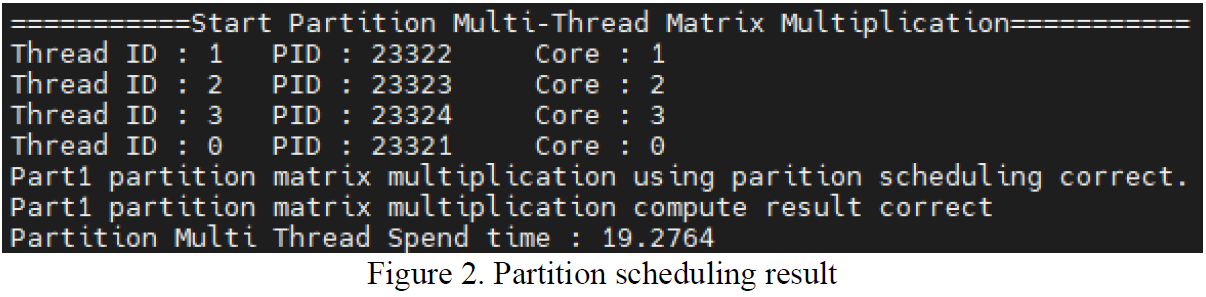
✓GNU品牌：4.1

第1部分：

將矩陣乘法劃分為可以同時執行的四個獨立部分。 然後將多線程執行與全局和分區調度配合使用，以提高矩陣乘法的性能。 執行結果如圖1所示。



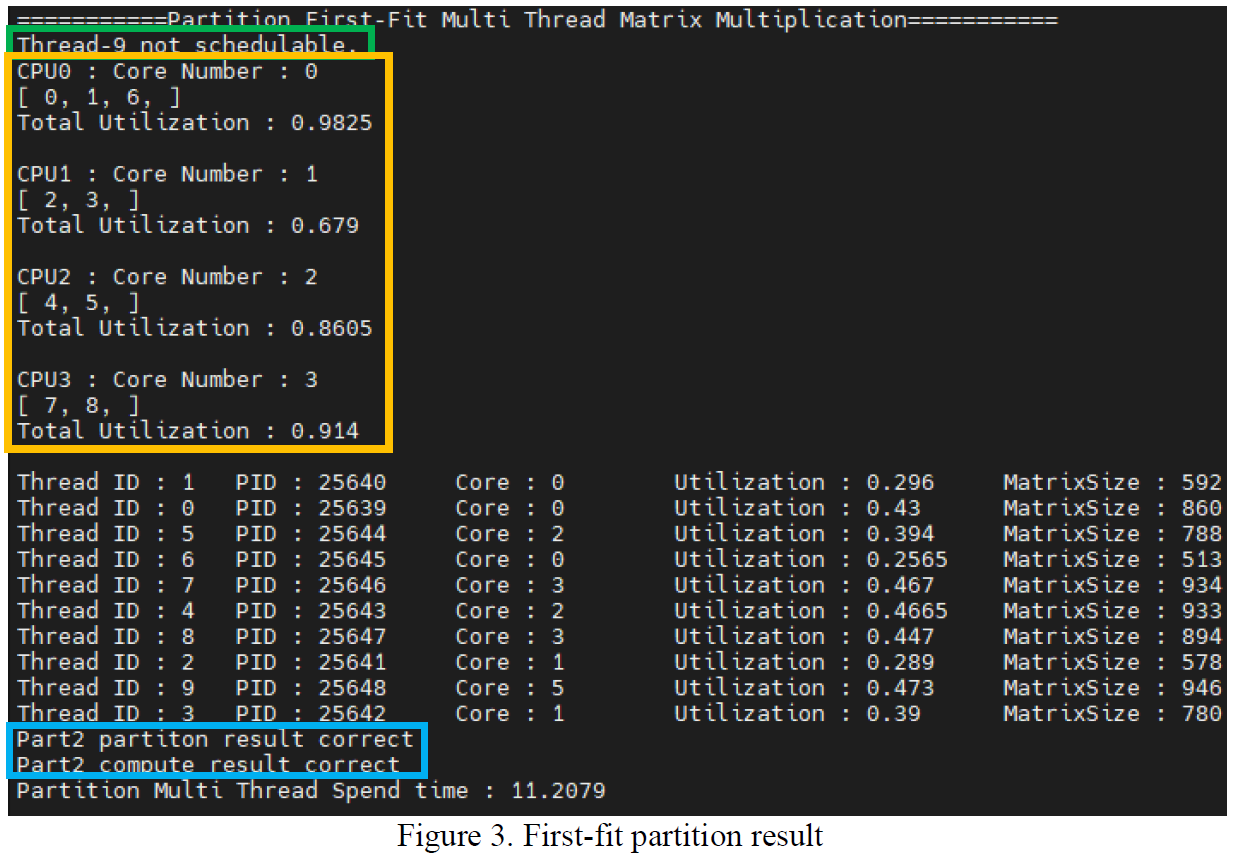
請按照圖1所示的格式打印全局多線程矩陣乘法的結果。首先，打印出帶有進程ID的線程信息，並在其上執行核心線程，並打印出每個線程的遷移狀態。 最後，使用類檢查（libs / check.h）打印出調度結果和矩陣乘法結果的正確性。



請按照圖2打印矩陣乘法的分區結果。

第2部分：

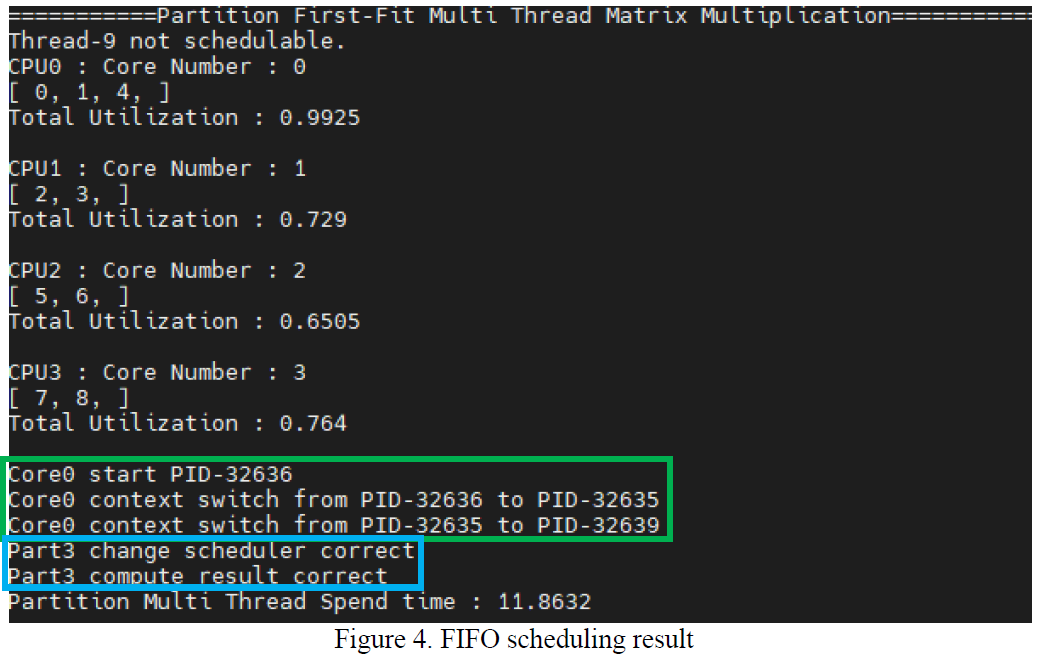
使用不同的分區方法（首次適應，最佳配合，最差配合）執行10次矩陣乘法（input / part2\_Input\_10.txt）和20次（input / part2\_Input\_20.txt）矩陣乘法。



請按照圖3所示的格式打印“最適合”，“最適合”和“最不適合”的分區結果。如果存在不可調度的線程，則以“ Thread-＃不可調度”格式打印出不可調度的線程 （如綠色框所示）。 否則，打印出分區結果（如黃色框所示）。 最後，如藍色框所示，使用類檢查（libs / check.h）打印出分區結果和矩陣乘法結果的正確性。

第3部分：

根據第2部分的分區結果使用FIFO和循環調度。



請按照綠色框中的格式在core-0上打印上下文切換狀態。 然後，使用類檢查（libs / check.h）打印出調度策略和矩陣乘法結果的正確性（如藍色框所示）。

命令行：

第1部分：

編譯：make part1.out

執行：./ part1.out part1\_Input.txt

第2部分：

編譯：make part2.out

執行：./part2.out part2\_Input\_10.txt ./part2.out part2\_Input\_20.txt

第3部分：

編譯：make part3\_rr.out make part3\_fifo.out

執行：sudo ./part3\_rr.out part3\_Input.txt sudo ./part3\_fifo.out part3\_Input.txt

預防措施

使用類變量“ pthreadThread”作為“ pthread\_create”的輸入參數。



記入：

⚫第一部分

[全局調度。 10％]

▪描述如何使用pthread實施全局調度。 5％

▪描述如何觀察任務遷移。 5％

[分區調度。 5％]

▪描述如何使用pthread實施分區調度。

[結果。 10％]

▪顯示任務的計劃狀態。 （您必須顯示使用輸入part1\_Input.txt的屏幕截圖結果）

⚫第2部分

[分區方法的實現。 10％]

▪描述如何在分區調度中實施三種不同的分區方法（先驗，最佳和最差）。

[結果。 30％]

▪顯示任務的計劃狀態。 （您必須顯示使用輸入part2\_Input\_10.txt和part2\_Input\_20.txt的屏幕截圖結果）

⚫第三部分

[計劃執行。 10％]

▪描述如何在分區調度中實現調度程序設置。 （帶FF的FIFO，帶FF的RR）

[結果。 10％]

▪顯示任務的流程執行狀態。 （您必須顯示使用輸入part3\_Input.txt的屏幕截圖結果）

⚫討論

▪在part1和part2中使用單線程和多線程分析和比較程序的響應時間。 （包括單身，全球，頭等，最佳，最差）10％

▪使用兩個不同的調度程序分析並比較程序的響應時間。 （帶FF的FIFO，帶FF的RR）5％

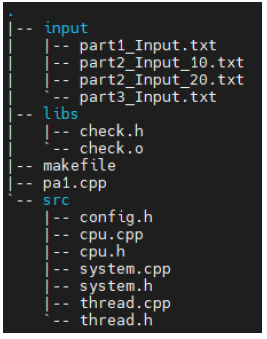
項目提交：

✓提交截止日期：4月12:30。 2021年21月

✓提交：Moodle

✓文件名格式：ESSD\_Student ID\_PA1.zip（例如ESSD\_M10704328\_PA1.zip）

✓包括源代碼：

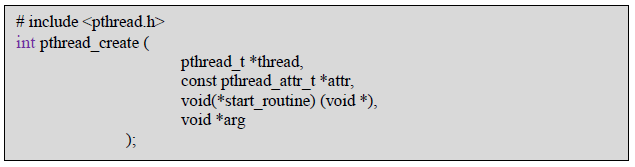


✓注意：ESSD\_Student ID\_PA1.zip必須包含報告和源代碼。

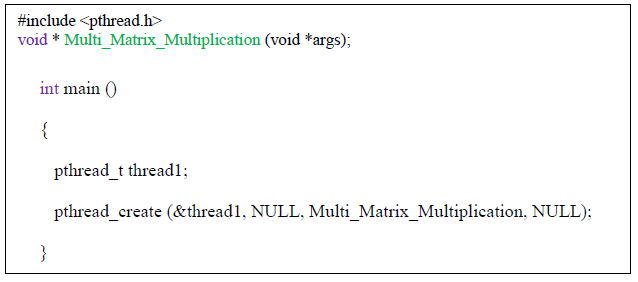
暗示：

POSIX線程創建

pthread\_create（）函數啟動一個新線程。

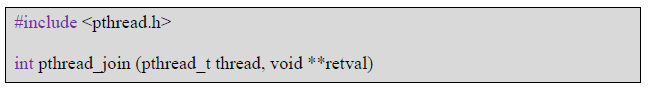


實現線程創建



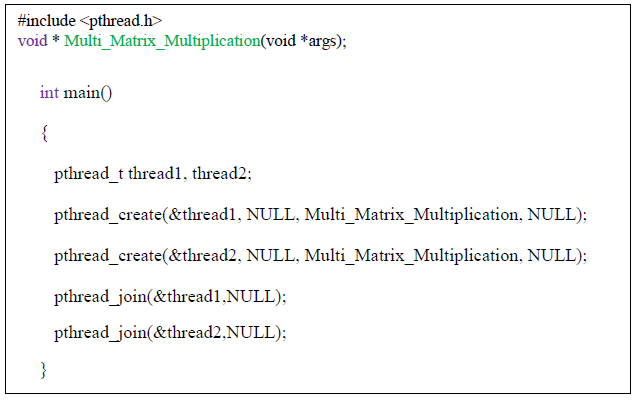
POSIX線程連接

函數pthread\_join（）允許調用線程等待目標線程的結束。 如果線程已經終止，則pthread\_join（）立即返回。 線程指定的線程必須是可連接的，這意味著該線程應終止。



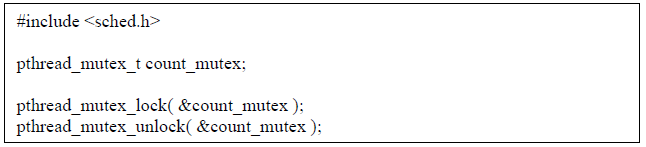
實現線程連接

當線程終止時，我們需要使用thread\_join（）來同步線程。 如果參數“ retval”不是Null，則pthread\_join（）將目標線程的退出狀態復製到“ retval”指向的位置。



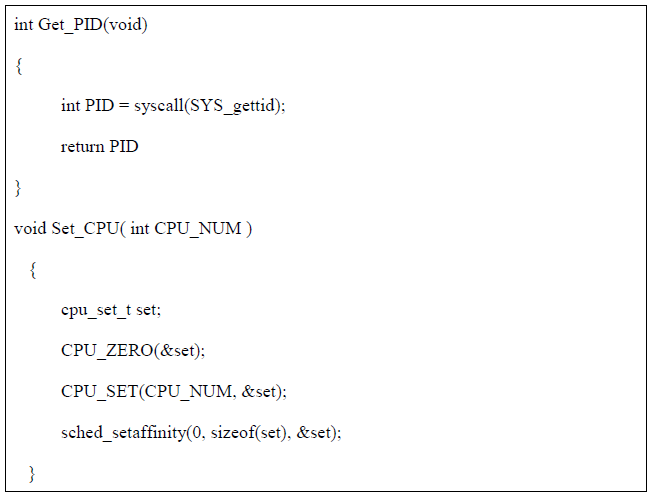
POSIX線程互斥量

可以通過調用pthread\_mutex\_lock（）來鎖定互斥對象。 如果互斥鎖已經被鎖定，則調用線程將阻塞，直到該互斥鎖可用為止。



系統調用

使用“ syscall（SYS\_gettid）”獲取當前線程的PID，並使用“ sched\_setaffinity（pid\_t pid，size\_t cpusetsize，const cpu\_set\_t \* mask）”來設置可以運行的CPU。



排程設定

Linux支持多種調度程序，例如FIFO和Round-Robin。 我們可以使用函數“ sched\_setscheduler（pid\_t pid，int policy，const struct sched\_param \* param）”將pid指定的進程的調度策略設置為policy，將調度參數設置為“ param”。

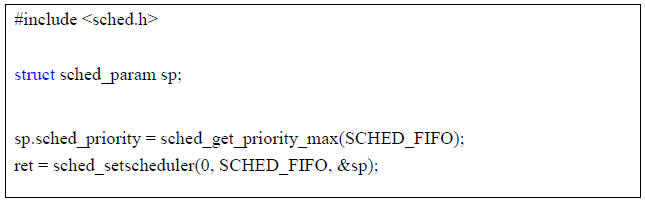
如果pid為0，則為調用線程設置策略和參數。 可以使用以下策略：

•SCHED\_FIFO

先進先出。 對於每個優先級，將按照將它們添加到要運行的進程隊列中的順序在CPU上執行進程。

•SCHED\_RR

輪循。 與SCHED\_FIFO相同，除了進程僅在定義的時間片上運行（請參閱sched\_rr\_get\_interval（））。 進程完成其時間片後，就其優先級而言，將其置於要運行的進程隊列的尾部。



Linux允許SCHED\_FIFO和SCHED\_RR的靜態優先級值為1到99。 請使用“ sched\_get\_priority\_max”設置進程的優先級，以使進程以“ RT”模式執行。