程式簡介

10727248 資訊三乙 鄭珮慈

一、開發平台

Intel® Core™ I5-8250U 處理器、RAM 8GB、64位元作業系統。

二、開發環境

Windows 10、CLion (IDE)。

三、程式設計說明

A、功能

1. 於程式開始執行時便提供使用者介面選項，可讓使用者決定要退出(QUIT)結束程式或是開始(START)執行程式。
2. 能夠依照使用者所輸入之檔案名稱讀檔，並依照檔案內所指定之排程MODE執行排程，最後輸出Gantt Chart與計算每個Process的Turnaround Time及Waiting Time。
3. 於排程結束時能夠將結果依標準格式寫檔輸出，且能夠重複執行(直到使用者於使用者介面選擇QUIT(退出)選項為止)。

→ 各個排程MODE運作說明：

(1). First Come First Served (FCFS)

依Arrival Time先後次序進行排程，當Arrival Time相同時，則依Process ID由小至大依序處理。

(2). Round Robin (RR)

依Arrival Time先後次序進行排序(時候未到的Process不能執行)，若有Arrival Time相同的情況，則依Process ID由小至大依序處理。

(a) 當Time Out時，從佇列尾端開始排序，若此時恰好有新來的Process，則讓新來的Process排在前面。

(b) 若Process的Time Slice未用完就結束，則必須讓下一個Process執行，且擁有完整的Time Slice。

(3). Shortest Remaining Time First (SRTF)

由剩餘CPU Burst最小的Process先排序，若剩餘的CPU Burst相同，則依Arrival Time小的先處理。若剩餘CPU Burst相同且Arrival Time相同，則依Process ID由小至大依序處理。

(4). Preemptive Priority + Round Robin (PPRR)

依Priority大小依序處理，Priority Number小的Process代表優先處理，若Priority相同的Process不只一個，則採用RR原則進行排程：

(a) 若有Priority相同的Process正在執行中，須等待其Time Slice用完。

(b) 當Time Out或被Preemptive時，從佇列尾端開始依Priority大小排序，若有新來的Process，則讓新來的Process排在前面。

(5). Highest Response Ratio Next (HRRN)

反應時間比率(Response Ratio)愈高的Process優先處理，若Ratio相同的Process不只一個，則依Arrival Time小的先處理。若Ratio相同且Arrival Time相同，則依Process ID由小至大依序處理。

(6). All

上述五種排程方法各執行一次。

B、流程

程式一開始，便會秀出使用者介面，使用者可選擇結束(QUIT)或開始執行(START)，若選擇START，就會請使用者輸入想進行排程的檔案，交由Method::ReadFile()讀檔。將資料都先儲存到vector(m_data)後，就會讓Method:: Schedule()進行排程，而在此function中，Method:: PreSortData()會先對儲存的資料進行Arrival Time的排序，倘若有Arrival Time相同者，就對他們進行Process ID的排序。

接著便開始依資料內所指定的排程方法對資料們進行排程。若method號碼為1，則執行Method:: FCFS()；號碼為2，則執行Method::RR()；號碼為3，則執行Method:: SRTF()；號碼為4，則執行Method:: PPRR()；號碼為5，則執行Method:: HRRN()；號碼為6，則執行Method:: All()。

排程完畢後便會用Method:: WriteFile()產出一個輸出檔，裡面存放的是排程好的資料及相關輸出資訊。

C、使用的Data Structure

我分別定義了兩種struct，分別叫fileContext (用於存原始資料與最終要輸出的結果資料，包含Process的Waiting Time和Turn Around Time)與IDInformation (用於記錄正在排程中的資料序列，含不斷變動的優先等級Response Ratio、Process的剩於CPU Burst和對應於原始資料中的位置)。詳細如下圖所示：

```
struct fileContext{
    int ID ;
    int CPU_Burst ;
    int Arrival_Time ;
    int Priority ;
    int TurnAroundTime ; // turnaround time = done time - arrival time
    int WaitingTime ;    // waiting time = turnaround time - CPU Burst
};
```

```
struct IDInformation{
    int ID ; // process ID
    float remain_CPUBurst ; // 剩餘的CPU Burst
    int Priority ;
    float ResponseRatio ; // = (waiting time + CPU Burst) / CPU Burst =>變動的優先等級(HRRN)
    int indexInData ; // 在m_data內的index
};
```

另外，於Method這個class內，我宣告了兩個array及五個vector，以下將針對此做說明。

第一個array的型別為int，負責暫存讀檔時每一行所切下來的數字們，由於每行最多只會有四個數字，因此大小設定為4。當一行中的四個數字都暫存到Num這個array時，就會以fileContext的形式存放資料形成一個block，再push_back進m_data這個vector，如此直到讀檔結束便完成輸入資料的暫存。

第二個array的型別為string，存放的是各個排程法的名稱。由於在寫所有排程法皆須執行的輸出檔時，想要以一個迴圈解決檔案上半部輸出排程名稱與對應甘特圖的部分，因此宣告了這個array，迴圈的index改變時，輸出的排程名稱也跟著變化。

m_data這個fileContext型式的vector，除了剛剛所說的在讀檔時暫存好所有輸入資料外，還會在每個Process執行完畢時，用Process ID找到它們在m_data內的空間存入他們的Waiting Time與Turn Around Time，最後輸出單一排程法之結果資訊時便以m_data此vector做輸出。

m_gantt為存放char型別的vector，會暫存目前執行排程法之甘特圖，每執行一個單位時間就應當會有一個Process ID表示字元(char)push_back存入，若當前單位時間無任何Process正在執行，則存入'-'此字元。

m_readyVector為IDInformation型式的vector，主要是跟存放原始資料的m_data做區隔，是一個Ready Queue的概念，裡面是已到達的Process們，正在排序等待執行。之所以不使用queue而用vector型別是因在排程過程中，我會需要對Process們進行排序，像是HRRN我就需要在過程中對Process們的Response Ratio做排序，PPRR也需要對Process們的Priority做排序。

m_AllGantt是為了存放所有排程法的甘特圖結果而創建的vector，是一個二維vector。當每個排程方法結束時，便會依照FCFS、RR、SRTF、PPRR、HRRN的順序將它們的甘特圖結果(vector)push_back進此vector，最後輸出檔案時便以此vector作所有排程法甘特圖的輸出。

m_AllData是為了存放所有排程法的結果資訊而創建的vector，是一個

二維vector。當每個排程方法結束時，便會依照FCFS、RR、SRTF、PPRR、HRRN的順序將它們的m_data(vector)資訊push_back進此vector，最後輸出檔案時便以此vector作所有排程法結果資訊的輸出。

詳細如下圖所示：

```
class Method {
private:
    int Num[4] ; // 暫存str內(四個)數字
    int currentTime = 0 ; // 目前時間
    string methodName[5] = { "FCFS", "  RR", "SRTF", "PPRR", "HRRN" };
    vector <fileContext> m_data ; // 資料們
    vector <char> m_gantt ; // 甘特圖
    vector <IDInformation> m_readyVector ; // 排隊等待運作的process們
    vector <vector <char>> m_AllGantt ; // 各個method的甘特圖
    vector <vector <fileContext>> m_AllData ; // 各個method資料結果們
```

四、不同排程法的比較

A、平均等待時間

【輸入檔案Input1.txt】15筆資料

FCFS：14.333

RR：18.4

SRTF：8.066 (最快)

PPRR：14.666

HRRN：11.6

【輸入檔案Input2.txt】5筆資料

FCFS：8.4

RR：6.4

SRTF：1 (最快)

PPRR：9.4

HRRN：8.2

【輸入檔案Input3.txt】6筆資料

FCFS：6.666 (最快)

RR：11.666

SRTF：6.666 (最快)

PPRR：12.5

HRRN：6.666 (最快)

B、結果與討論

由結果得知利用SRTF的平均等待時間會最短，其他則是會受輸入資料或是資料筆數影響而有不同運作效果。