

# Tiny Scheduler（微型排程器）

在這個作業中，你將用 C 語言實作一個 CPU 排程模擬器，展示以下排程演算法的行為：

- 先到先服務（FCFS）；非搶佔式
- 時間片輪轉（RR）：搶佔式

目標是分析不同排程策略如何影響程序執行順序、回應時間和周轉時間。

**注意：這只是模擬器，不需要建立真正的程序。**

## 閱讀

本專案涵蓋教科書第 7 章和課程教材第 2 講的內容。你的程式碼必須有適當的註解。請參考評分標準並閱讀 `code_style` 指南以了解註解要求。

在閱讀完整份說明並理解測試案例前，請勿開始寫程式碼。

## 概述

對於這個微型排程器，假設所有程序只使用 CPU（沒有 I/O 操作）。每個程序的執行時間事先已知。假設所有程序的到達時間都不同，但執行時間可以相同或不同。

請記住，作業系統始終維護一個就緒佇列。當新程序到達時，會被加入就緒佇列。你應該為就緒佇列實作一個佇列。**\*\*到達時間不是事先已知，而是在模擬時間從 0 開始的「即時」發生。\***正確的實作應該在程序到達時將其加入佇列，並從佇列中處理。時間從 0 開始，只有在程序到達時才加入佇列。

對於時間片輪轉（RR），當發生上下文切換時，目前執行的程序會被放回就緒佇列的末端，形成循環。然而，FCFS 不會這樣做，因為它會將程序執行到結束。

你的程式應該這樣執行：`./tsim fcfs|rr input_filename [time_slice]`

`input_filename` → 指定包含任務列表的檔案。

`fcfs` 或 `rr` → 指定排程策略。

`time_slice` → 僅在 RR 時需要。

你的程式應從文字檔讀取輸入資料（提供了三個範例輸入檔）。每個檔案的格式為：PID Arrival\_Time Run\_Time

PID → 字串。唯一的數字程序識別碼。

Arrival\_Time → 整數。程序到達的時間（毫秒）。

Run\_Time → 整數。所需的 CPU 執行時間（毫秒）。

**程序在輸入檔中總是依到達順序排序。**

你的程式應以結構化格式顯示排程過程（請參考本頁底部的測試案例），顯示時間區間、目前執行的程序，以及就緒佇列中的程序。FCFS 每毫秒顯示一次，RR 則以每個時間片或剩餘執行時間（取較小者）為單位。最後，請輸出平均回應時間、平均周轉時間和上下文切換次數。

**錯誤處理**

部分命令列參數驗證由 `tester.c` 處理，但 `sim.c` 還需實作以下檢查：

1. 檢查 RR 是否有提供時間片，且範圍在 **2-4 毫秒（含）**。
2. 驗證輸入檔是否存在。
3. 驗證讀取檔案後程序數量是否介於 2 到 5 之間。
4. 可以假設執行時間不會為 0，這部分不需檢查。

對於命令列參數錯誤（1 和 2），請呼叫 `tester.c` 的 `usage_message()` 函數並給出適當錯誤訊息。其他錯誤請提供清楚且具體的錯誤訊息。

### 起始程式碼：

下載 `tsim.zip` 並先瀏覽提供的程式碼。

`/tsim`

— Makefile	// 請勿更改。
— tester.c	// 請勿更改。處理命令列參數並呼叫排程函數。
— utils.h	// 請勿更改。定義結構、常數和函數原型
— input1.txt	// 範例輸入檔
— input2.txt	// 範例輸入檔
— input3.txt	// 範例輸入檔

### 提交

**\*\*只需提交 `sim.c` 檔案；\*\*請勿提交壓縮檔。**

你的程式會用原始的 `Makefile`、標頭檔、文字檔和 `tester.c` 編譯。

**\*\*請勿更改任何提供的檔案，\*\*否則可能導致編譯失敗。**

**測試** [請先手動驗證測試案例再開始寫程式]

**./tsim fcfs input1.txt**

Running FCFS

Time | Running | Ready Queue

0 - 1	P1	-
1 - 2	P1	P2
2 - 3	P1	P2 P3
3 - 4	P1	P2 P3
4 - 5	P1	P2 P3 P4
5 - 6	P2	P3 P4 P5
6 - 7	P2	P3 P4 P5
7 - 8	P2	P3 P4 P5
8 - 9	P3	P4 P5
9 - 10	P3	P4 P5
10 - 11	P3	P4 P5
11 - 12	P3	P4 P5
12 - 13	P4	P5
13 - 14	P4	P5
14 - 15	P5	-
15 - 16	P5	-
16 - 17	P5	-
17 - 18	P5	-
18 - 19	P5	-
19 - 20	P5	-

FCFS Execution Completed.

Average Response Time: 4.60 ms

Average Turnaround Time: 8.60 ms

Total Context Switches: 4

**./tsim fcfs input2.txt**

Running FCFS

Time | Running | Ready Queue

0 - 1	IDLE	P1
1 - 2	P1	P2
2 - 3	P1	P2 P3
3 - 4	P1	P2 P3
4 - 5	P1	P2 P3
5 - 6	P1	P2 P3
6 - 7	P2	P3
7 - 8	P2	P3
8 - 9	P2	P3
9 - 10	P3	-
10 - 11	IDLE	-
11 - 12	IDLE	P4
12 - 13	P4	-

FCFS Execution Completed.  
Average Response Time: 2.5ms  
Average Turnaround Time: 5 ms  
Total Context Switches: 3

```
./tsim fcfs input3.txt
```

Running FCFS  
Time | Running | Ready Queue

0 - 1	P1	-
1 - 2	P1	P2
2 - 3	P1	P2
3 - 4	P1	P2 P3
4 - 5	P1	P2 P3
5 - 6	P1	P2 P3
6 - 7	P1	P2 P3
7 - 8	P1	P2 P3
8 - 9	P2	P3
9 - 10	P2	P3 P4
10 - 11	P2	P3 P4
11 - 12	P3	P4
12 - 13	P3	P4
13 - 14	P3	P4
14 - 15	P3	P4

15 - 16	P3	P4
16 - 17	P4	-
17 - 18	P4	-

FCFS Execution Completed.  
Average Response Time: 4.75 ms  
Average Turnaround Time: 9.25 ms  
Total Context Switches: 3

```
./tsim rr input3.txt 2
```

Running RR  
Time | Running | Ready Queue

-----		
0 - 2	P1	P2
2 - 4	P2	P1 P3
4 - 6	P1	P3 P2
6 - 8	P3	P2 P1
8 - 9	P2	P1 P3
9 - 11	P1	P3 P4
11 - 13	P3	P4 P1
13 - 15	P4	P1 P3
15 - 17	P1	P3
17 - 18	P3	-

RR Execution Completed.  
Average Response Time: 1.25 ms  
Average Turnaround Time: 10.75 ms  
Total Context Switches: 9

```
./tsim rr input3.txt 4
```

Running RR  
Time | Running | Ready Queue

-----		
0 - 4	P1	P2 P3
4 - 7	P2	P3 P1
7 - 11	P3	P1 P4
11 - 15	P1	P4 P3

15 - 17	P4	P3
17 - 18	P3	-

RR Execution Completed.

Average Response Time: 2.5 ms

Average Turnaround Time: 10.25 ms

Total Context Switches: 5

## 常見問題

### 1. FCFS 和 RR 可以用同一個循環陣列資料結構嗎？

可以。FCFS 執行完畢後不會重新插入程序，不會有影響。你可以用陣列型佇列（處理環狀）或鏈結串列型佇列，維護 front 和 rear 指標。

### 2. 執行中的程序要出現在就緒佇列嗎？

執行中的程序不應出現在佇列。就緒佇列只包含等待中的程序，不包含正在執行的。如果執行中的程序還在佇列，可能會造成重複排程。

但課堂上為了視覺化，有時會將執行中的程序顯示在佇列。這主要是為了方便觀察，實作時不應這樣做。