開發環境

本程式在 Windows 64-bit 環境下開發，使用 C++ 語言編寫。

實作方法和流程

A. 印出主選單，等待使用者輸入檔案名稱。

B. 嘗試以檔案名稱開啟檔案。
   a. 若開檔失敗，則回到 1.。
   b. 若開檔成功，則呼叫 readFile() 將檔案資料讀入。

C. 根據檔案中選擇的排序方式來執行 FCFS()、RR()、SJF()、SRTF()、HRRN()、PPRR() 中的其中一個或全部一起執行。

D. 將各排程法的結果透過 PrintSchedule() 輸出至檔案中。

E. 回到步驟 1，等待下一次 User 輸入。

- **FCFS()：**
  1. 呼叫 PushArrivalProcessInReadyQueue() 將已抵達的 process 加到 ReadyQueue。
     a. 若 ReadyQueue 為空，將 GanttChart 加上 "-" 代表此時間點上沒有 process 被執行。time++，回到 1.。
     b. 若 ReadyQueue 不為空，取出其中的第一個 process。
        將它的 status 設為 "Execution"，並從 ReadyQueue 中移除它。

  2. 對於該 process，用 for 迴圈模擬 CPU 執行它的 CPU_burst 單位時間。
     在每個時間點上：
     在 GanttChart 加上它的代號。
     呼叫 PushArrivalProcessInReadyQueue() 把當前已經到達的 process 加到 ReadyQueue 中。
     ReadyQueue 中的每個 process 的 waitingTime++。

     在 for 迴圈結束後，代表此 process 已經完成，將它的 status 設為 "Done"，並計算 turnaroundTime。

  3. 回到 1.，重複以上步驟，直到所有 process 執行完畢為止。

- RR() :
    1. 呼叫 PushArrivalProcessInReadyQueue() 將已抵達的 process 加到 ReadyQueue。
        a. 若 ReadyQueue 為空，將 GanttChart 加上 "-" 代表此時間點上沒有 process 被執行。time++，回到 1.。
        b. 若 ReadyQueue 不為空，取出其中的第一個 process。
           將它的 status 設為 "Execution"，並從 ReadyQueue 中移除。

    2. 對於該 process，用 for 迴圈模擬 CPU 執行 timeSlice 單位時間。
       在每個時間點上：
           在 GanttChart 加上它的代號，CPU_burst--。
           呼叫 PushArrivalProcessInReadyQueue() 把當前已經到達的 process 加到 ReadyQueue 中。
           把 ReadyQueue 中的每個 process 的 waitingTime++。

       在 for 迴圈結束後，
        a. 若該 process 的 CPU_burst 為 0，代表它已經完成，將它的 status 設為 "Done"，並計算 turnaroundTime。
        b. 若該 process 的 CPU_burst 不為 0，代表它未完成，將它的 status 設為 "Not Done Yet"，並加回 ReadyQueue 的尾端。

    3. 回到 1.，重複以上步驟，直到所有 process 執行完畢。

- SJF() :
    1. 呼叫 PushArrivalProcessInReadyQueue() 將已抵達的 process 加到 ReadyQueue。
        a. 若 ReadyQueue 為空，將 GanttChart 加上 "-" 代表此時間點上沒有 process 被執行。time++，回到 1.。
        b. 若 ReadyQueue 不為空，取出其中 CPU burst 最短的 process。
           將它的 status 設為 "Execution"，並從 ReadyQueue 中移除。

    2. 對於該 process，用 for 迴圈模擬 CPU 執行它的 CPU_burst 單位時間。
       在每個時間點上：
           將 GanttChart 加上它的代號。
           呼叫 PushArrivalProcessInReadyQueue() 把當前已經到達的 process 加到 ReadyQueue 中。
           把 ReadyQueue 中的每個 process 的 waitingTime++。

       在 for 迴圈結束後，代表該 process 已經完成，將它的 status 設為 "Done"，並計算 turnaroundTime。

    3. 回到 1.，重複以上步驟，直到所有 process 執行完畢。

- SRTF()：
    1. 呼叫 PushArrivalProcessInReadyQueue() 將已抵達的 process 加到 ReadyQueue。
        a. 若 ReadyQueue 為空，將 GanttChart 加上 "-" 代表此時間點上沒有 process 被執行。time++，回到 1.。
        b. 若 ReadyQueue 不為空，取出其中 CPU burst 最短、抵達時間最早、pid 最小的 process。
        將它的 status 設為 "Execution"，並從 ReadyQueue 中移除。

    2. 執行該 process 一個時間單位：
        在 GanttChart 加上它的代號，CPU_burst--。
        將 ReadyQueue 中所有 process 的 waitingTime++。

        a. 若該 process 的 CPU_burst 為 0，代表它已經完成，將它的 status 設為 "Done"，並計算 turnaroundTime。
        b. 若該 process 的 CPU_burst 不為 0，代表它未完成，將它的 status 設為 "Not Done Yet"，並加回 ReadyQueue 的尾端。

    3. 回到 1.，重複以上步驟，直到所有 process 執行完畢。

- HRRN()：
    1. 呼叫 PushArrivalProcessInReadyQueue() 將已抵達的 process 加到 ReadyQueue。
        a. 若 ReadyQueue 為空，將 GanttChart 加上 "-" 代表此時間點上沒有 process 被執行。time++，回到 1.。
        b. 若 ReadyQueue 不為空，計算 ReadyQueue 中每個 process 的 ResponseRatio ，並取出其中 ResponseRatio 最高的 process。
        將它的 status 設為 "Execution"，並從 ReadyQueue 中移除。

    2. 對於該 process，用 for 迴圈模擬 CPU 執行它的 CPU_burst 單位時間。
    在每個時間點上：
        將 GanttChart 加上它的代號。
        呼叫 PushArrivalProcessInReadyQueue() 把當前已經到達的 process 加到 ReadyQueue 中。
        把 ReadyQueue 中的每個 process 的 waitingTime++。

    在 for 迴圈結束後，代表該 process 已經完成，將它的 status 設為 "Done"，並計算 turnaroundTime。

    3. 回到 1.，重複以上步驟，直到所有 process 執行完畢。

- PPRR()：

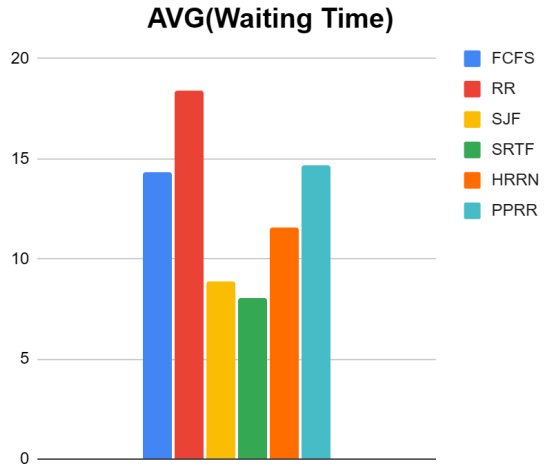
1. 呼叫 PushArrivalProcessInReadyQueue() 將已抵達的 process 加到 ReadyQueue。
   呼叫 SortReadyQueueByPriority() 將 ReadyQueue 中的 process 進行排序。
   a. 若 ReadyQueue 為空，將 GanttChart 加上 "-" 代表此時間點上沒有 process 被執行。time++，回到 1.。
   b. 若 ReadyQueue 不為空，取出其中的第一個 process。
      將它的 status 設為 "Execution"，並從 ReadyQueue 中移除。

2. 對於該 process，用 for 迴圈模擬 CPU 執行 timeSlice 單位時間。
   在每個時間點上：
   在 GanttChart 加上它的代號，CPU_burst--。
   呼叫 PushArrivalProcessInReadyQueue() 把當前已經到達的 process 加到 ReadyQueue 中。
   呼叫 SortReadyQueueByPriority() 將 ReadyQueue 中的 process 進行排序。
   把 ReadyQueue 中的每個 process 的 waitingTime++。
   若剛加入的 process 的 priority 比此 process 還高的話，跳至 3.

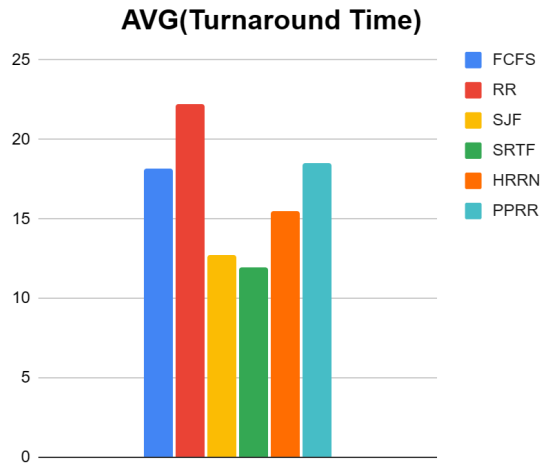
   在 for 迴圈結束或該 process 的 CPU_burst 為 0 後，
   a. 若該 process 的 CPU_burst 為 0，代表它已經完成，將它的 status 設為 "Done"，並計算 turnaroundTime。
   b. 若該 process 的 CPU_burst 不為 0，代表它未完成，將它的 status 設為 "Not Done Yet"，並加回 ReadyQueue 的尾端。

3. 回到 1.，重複以上步驟，直到所有 process 執行完畢。

不同排程法的比較



圖表 1、各排程法之平均 Waiting Time



圖表 2、各排程法之平均 Turnaround Time

在本次作業中，我們實作了 FCFS、RR、SJF、SRTF、HRRN 和 PPRR 六種排程演算法。接下來，將針對這六種演算法的優點和缺點進行討論。

FCFS 演算法的優點在於簡單易懂，容易實作。它的缺點在於平均等待時間較長，因為當執行時間長的 process 進入 CPU 時，後面排隊的 process 就必須等待很長時間才能執行。因此，在執行時間較長的 process 進入系統時，FCFS 的效能會降低。

RR 演算法的優點在於每個 process 都有機會被執行，避免了某些 process 長時間等待 CPU 的情況。它的缺點在於 time slice 長度的設定會影響效能，若 time slice 過短，會造成 context switch 的次數增加，而且可能會浪費 CPU 資源；若 time slice 過長，則會導致平均等待時間增加。

SJF 演算法的優點在於能夠減少平均等待時間，因為優先執行執行時間短的 process，讓執行時間長的 process 進入 CPU 前，先讓執行時間短的 process 優先執行完畢。
但它的缺點在於當有新的 process 進入系統時，需要重新評估所有 process 的執行時間，這會造成一定的開銷。

SRTF 演算法是 SJF 的變形，它的優點在於能夠進一步縮短平均等待時間，因為它考慮了新進入的 process 的執行時間，能夠更加靈活地調度 CPU，但它的缺點和 SJF 一樣會在評估 process 的執行時間上造成一定的開銷。

HRRN 演算法是一種動態的優先級調度算法，它考慮了 process 的等待時間和執行時間，以計算出 process 的優先級。這種演算法的優點在於能夠避免 process 長時間等待 CPU 的情況，但它的缺點在於計算優先級需要耗費額外的計算資源。

PPRR 演算法是一種動態的優先級調度算法，。它的優點在於能夠更好地適應不同的應用場景，但是它的缺點在於需要額外的計算資源和更複雜的調度策略。