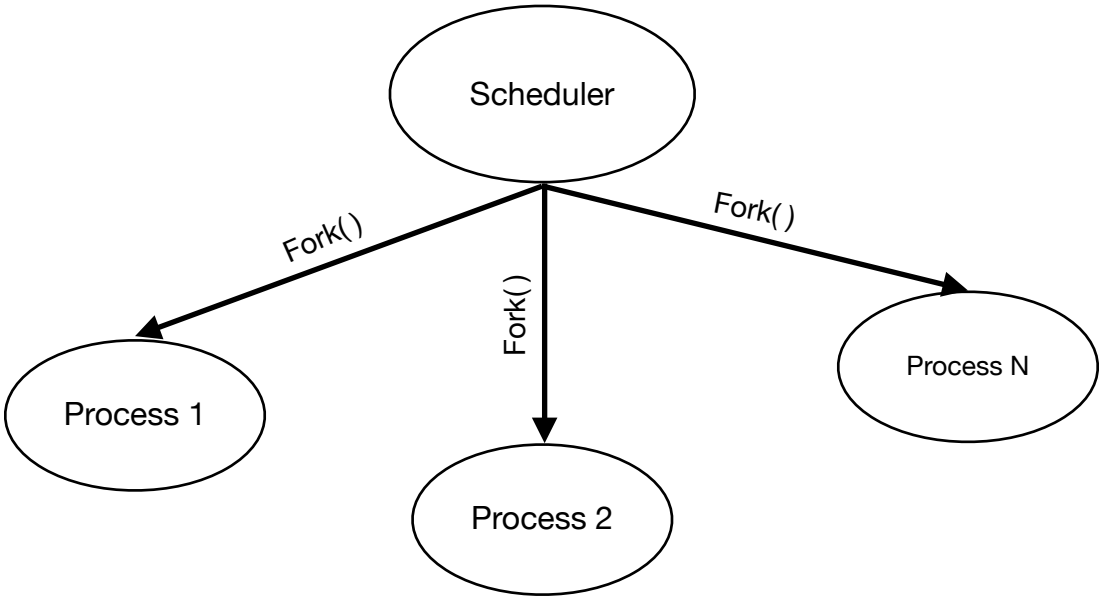
B05501024

土木三 黃政維

Project 1

-1. 設計

這支程式主要由 main 進入，接著執行 scheduler ， scheduler 為執行主要程式的部分，負責讀取輸入之資料，生成 process 並且管控 process 使用 cpu 的時間，如圖一所示。



圖一

Scheduler 會先在 readfile 的過程中生成新的 processInfo ，其內容存取 process 的 name 、 ready time 、 execution time 、 origin execution time 、 pid 需要特別說明的是 execution time 會隨著 process 真正執行而遞減， origin execution time 則是會真正紀錄原始的 execution time ，因此根據這兩個值相減即可得到 process 所執行的時間。所有的 processInfo 皆是使用動態記憶體配置的，並且將所有生成的 processInfo 的 pointer 儲存在 processList 當中。 processInfo 如圖二

所示。並且 Scheduler 會在讀取完 input 後先對 processList 裡的 pointer 依照 ready time 進行排序。

```
12  typedef struct processInfo
13  {
14      char name[33];
15      int  ready_time   ;
16      int  execution_time;
17      int  origin_execution_time;
18      pid_t pid    ;
19  }processInfo;
```

圖二

各個排程方法的實踐其實大同小異，我設計了一個 ready queue，其作用在於等待被執行，所有 process 到了 ready time 時就會進入這個 ready queue 等待被執行，並且這個 ready queue 為一個 bounded queue ，大小為 number of process ，因為相同的process 不可能同時在 ready queue 裡出現，因此其大小不可能超過 number of process ，所以當 ready queue 裡等待的 process number 大於 number of process 時就代表出錯，並且中斷程式。 ready queue 就是我自己設計的 processQueue ，其內容有 processes ，其為存取進入 ready queue 的 process 的 pointer，in 為下一個可以放置 process pointer 至 queue 的位置， out 為下一個可以取出的位置，num 為當下 queue 內 pointer 的數量， maxNum 為 queue 可以填入 pointer 數量之上限，如圖三所示。

```
7   typedef struct processQueue
8   {
9       struct processInfo** processes;
10      int    in     ;
11      int    out    ;
12      int    num    ;
13      int    maxNum ;
14  }processQueue;
```

圖三

接下來就各個排程機制詳細解說。

FIFO：

在 FIFO 中，當有 process 到達其 ready time 時就會被 push 進入 ready queue 中，而當目前沒有 process 正在執行的話， scheduler 就會去 ready queue 內 pull 最上層的 process 出來執行，若 process 尚未被 fork 出來就由 scheduler 使用 fork( ) 生成子進程。

RR：

和 FIFO 一樣，當 process 到達 ready time 時就會被推進 ready queue 內，不同的是，若當前正有 process 正在執行的話，便會去計算 process 是否已執行五百個單位了，計算方法由前述的 execution time 以及 origin execution time 相減計算，若執行至五百個單位，當前 process 就會被推進 ready queue 當中，然後再從 、ready queue 之中拉取新的 process 執行。

SJF:

前述兩種方式在將 process 放入 queue 之中的方法是採用 first in first out 的方法，而 SJF 當中則採用 insert 的方法，每當有新的 process 要進入 queue 當中的

時候，就會從 queue 的頭開始往後一個個在 ready queue 中等待的 process 用 execution time 進行比較，若欲進入 ready queue 的 process 的 execution time 比較小則插入。

PSJF：

把 process 推入 queue 的方式和 PSJF 的方式一樣，只是如果當前有 process 正在執行，則會不斷檢查 ready queue 的頭個 process 的 execution time 是否比當前執行的 process 的 execution time 小，若小於當前執行的 process 則將當前的 process 插入 ready queue 。

不論哪種排程機制皆會在沒有 process 正在執行時從 ready queue 中拉出 process 執行，並且在有 process 完成後記錄下來， 當完成的 process 數量等於所有 process 的數目後即結束排成作業。所以各個排成的差別只在於進入 ready queue 的方式以及是否會被中斷而已，其方式如上述，並且 scheduler 以及 process 各使用一顆 cpu。

-2. 核心版本

本次測試之核心版本為作業一所教的 linux-4.14.25。

-3. 比較實際結果與理論結果差異

就各個 process 結束時間上的順序，以及 scheduler 的排程過程我認為我實現得差不多，但因為自身時間上的原因，並沒有特別探討實際與理論上之實際差異，但我認為如下的事情可能發生並且導致成果有差異。

我認為 FIFO 以及 SJF 的 process 在執行時間上可能不會有太多的誤差，因為基本上 parent process 皆會等待 child process 結束後再生成下一個 child process，可能造成較大的誤差可能為 RR 以及 PSJF 兩個部分，因為當我在判斷是否需要將當前 process 推入 ready queue 的時候，child process 仍舊在運行中，如下假設，當 scheduler 預計在第二秒時暫停 process 運行並且推入 ready queue 當中，理論 process 也應該在自己執行到第二秒時被其他 process 佔用 cpu ，但實際上 process 可能仍舊運行到二點二秒才被終止，而此誤差可能在多個 process 輪流搶用 cpu 的時候產生最大的誤差，因此我猜測在 RR 排程的時候會發生最大的誤差。