OS Project2 說明文件

資訊三乙 10927256 姜美羚

一 、開發環境:

作業系統 : win10
IDE : Visual Studio Code
開發語言: C++
gcc version : 6.3.0 (MinGW.org GCC-6.3.0-1)

二 、實作方法&流程:

資料結構:
struct node ; => 用於存放單一 Process 之所有資訊，如:PID、Arrival time….
vector<node> pList ; => 用於存放從 input 中讀取的所有要做的 Process
vector<vector<node>> fList ; =>是一個二維動態陣列，用於存放使用各個方法排程之結果。如: 每一個 Process 的 waitting time、Turnaround time。
vector<vector<node>> gantte ; =>是一個二維動態陣列，用於存放使用各個方法排程之甘特圖。

前置作業:

將讀入的資料，依照 Arrival time 排序由小到大進行排序，若 Arrival time 相同，則依照 PID 由小到大進行排序。以模擬 Process 先來後到的情況。

方法 1(FCFS):

Process 依照 Arrival time，進入 readyQ 中。每次從 readyQ 的最前面，取一個 Process 放入 CPU 中執行。因為 FCFS 是不可奪取的，所以當完整做完一個 Process 後，才進行 context switch。當發生 context switch 時，從 readyQ 中取下一個要執行的工作放到 CPU 中執行。直到所有已抵達的 Process 都完成工作，則終止排程工作並結束。

方法 2(RR):

Process 依照 Arrival time，進入 readyQ 中。每次從 readyQ 的最前面，取一個 Process 放入 CPU 中執行。當 Process 進入 CPU 時，使用 intoTime 紀錄 Process 進入 CPU 的時間。當正在 CPU 執行的 Process 發生 timeout 時，如果有新的 Process 剛好來，則先放入新來的 Process，再將被換下來的 Process 放入 readyQ 中。若 Process 結束工作，且並未使用完一個完整的 timeslice，則直接從 readyQ 中，取下一個要做的 Process，並使其擁有一個完整的 timeslice。直到所有已抵達的 Process 都完成工作，則終止排程工作並結束。

方法 3(SJF):

Process 依照 Arrival time，進入 readyQ 中，並依照 CPU Brust time 由小到大排序。每次從 readyQ 的最前面，取一個 Process 放入 CPU 中執行。因為 SJF 是不可奪取的，所以當完整做完一個 Process 後，才進行 context switch。當發生 context switch 時，從 readyQ 中取下一個要執行的工作放到 CPU 中執行。直到所有已抵達的 Process 都完成工作，則終止排程工作並結束。

方法 4(SRTF):

Process 依照 Arrival time，進入 readyQ 中。每次從 readyQ 中，找出剩餘 CPU Brust time 最小的 Process，放入 CPU 中執行。若剩餘 CPU Brust time 相同，則找 Arrival time 較小的優先執行。若 Arrival time 相同，則找 PID 較小的優先執行。因為 SRTF 是可奪取的，所以當有新的 Process 抵達時，就要從目前所有已到達的 Process 中，找出剩餘 CPU Brust time 最小的優先執行。而當前正在做的 Process 被 Preemptive 時，要將被 Preemptive 的 Process 再放回 readyQ 中。直到所有已抵達的 Process 都完成工作，則終止排程工作並結束。

方法 5(HRRN):

Process 依照 Arrival time，進入 readyQ 中。每次從 readyQ 中，找出 Response ratio 最小的 Process，放入 CPU 中執行。若 Response ratio 相同，則找 Arrival time 較小的優先執行。若 Arrival time 相同，則找 PID 較小的優先執行。因為 HRRN 是不可奪取的，所以當完整做完一個 Process 後，才進行 context switch。直到所有已抵達的 Process 都完成工作，則終止排程工作並結束。

#Response ratio 計算公式:

Process_wait = curTime – Process_Arrivaltime

Response ratio = (Process_wait + Process_CPU Brust time) / Process_CPU Brust time

方法 6(PPRR):

Process 依照 Arrival time，進入 readyQ 中，並依照 Priority 由小到大排序。每次從 readyQ 中，找出 Priority 最小的 Process，放入 CPU 中執行。若目前已到達的所有 Process 的 Priority 都相同，則改採用 RR 進行排程。因為 PPRR 是可奪取的，因為每當有新的 Process 抵達時，都要檢查是否有比目前正在執行的 Process 的 Priority 更小的 Process 出現。直到所有已抵達的 Process 都完成工作，則終止排程工作並結束。

三 、不同排程法的比較:

	FCFS	RR	SJF	SRTF	HRRN	PPRR
input1	14.33333	18.4	8.866667	8.066667	11.6	14.66667
input2	8.4	6.4	8.2	3	8.2	9.4
input3	6.666667	11.66667	6.666667	6.666667	6.666667	12.5
input4	3.75	5.5	3.5	3.25	3.75	4.5

表 1、6 個排程法平均等待時間(Average Waiting Time)比較

	FCFS	RR	SJF	SRTF	HRRN	PPRR
input1	18.2	22.26667	12.73333	11.93333	15.46667	18.53333
input2	13.2	11.2	13	7.8	13	14.2
input3	24.16667	29.16667	24.16667	24.16667	24.16667	30
input4	8.75	10.5	8.5	8.25	8.75	9.5

表 2、6 個排程法平均往返時間(Average Turnaround Time)比較

以上是使用公告的四個測資(input1~ input4)所繪製的圖表。

由表 1 和表 2 可以看出，在大多數的情況下，使用 SRTF 進行排程，平均等待時間和平均往返時間都耗時最短，而使用 RR 進行排程，平均等待時間和平均往返時間都耗時最長。雖然 SRTF 看起來是在這些方法之中，效率最好的，但是如果出現 CPU Burst time 較大的 Process，可能會造成 starvation 的狀況，造成 Process 無限期的等待。而且也會導致 SRTF 的 2 個平均時間都增加。

而 RR 排程法，因為在公告的四個測資中，timeslice 的範圍落在 1~3 之間，但如果將 timeslice 調大，甚至是超過所有 Process 的 CPU Burst time 時，使用 RR 排程法的結果就會和 FCFS 相等，如表 3.。因為並不會有 Preemptive 的狀況發生，每次都是完整的做完一個 Process 才進行 context switch。

timeslice	1	3	100
input1	18.4	16.27	18.2
input2	6.6	6.4	13.2
input3	13.17	13.6667	24.16667

表 3、RR 排程法使用不同 timeslice 之平均等待時間(Average Waiting Time)比較

四 、結果與討論:

經由實作各種排程法，並觀察模擬結果。我認為排程法並沒有優劣之分，每個排程法都有各自的優缺點。我覺得更重要地反而是，針對不同的情況、需求，去選擇較合適的排程法，才能提高效率。

而在這些排程法中，我覺得 PPRR 是相對起來，在這之中比較公平且較符合實務需求的排程法。因為如同現實生活中，每件事都有其緊急程度和重要性之程度差異。如同排程法的設計，優先做 Priority 較小的，對應到的就是生活中遇到較為緊急的事件，應該要優先處理。而當 Priority 相同時，則改採 RR 進行排程，用較公平的方式，去輪流處理，平均的去完成緊急程度一樣的工作，這種設計方式比較符合生活中有效率處理事情方式。