OS Project1 Report

b07902117 資工二 陳漱宇

1. 設計

首先，讀完 $input$ 後跑 $scheduler()$，$scheduler()$ 的工作在於決定哪個 $process$ 可以占用 $core$，並調整 $process$ 的 $priority$ 使其佔有或不佔有 $core$，並且我將 $core$ 0 給 $scheduler$ 的 $process$ 跑，將 $core$ 1 給其他 $process$ 跑，使其平行化進行。

$scheduler()$ 中首先根據 $process$ 的 $ready\ time$ 從小排到大，接著跑 $while$ 迴圈直至所有 $process$ 皆完成，$while$ 迴圈中檢查現在是否有 $process$ 剛好 $ready$ 了，有則 $fork$ 一個 $child\ process$ 給它 ($new\_proc\_exec()$) 並降低它的 $priority$ ($block\_proc()$) (只是先建好 $process$)。再來檢查當前的 $process$ 是否已經完成，若是，則用 $waitpid()$ 來收屍，並將完成的數量 $+1$。接著選擇下一個時間點輪到哪個 $process$ 跑 ($choose\_next()$)，並把當前正在跑的 $process$ 的 $priority$ 降低，把下個時間點輪到的 $process$ 的 $priority$ 升高。最後 $scheduler()$ 跑 $a\ unit\ of\ time$ (為了跟 $process$ 同時進行) 並把當前時間 $+1$。

在 $choose\_next()$ 中，我分別討論四種可能的 $scheduling\ policy$:

1. $first\ in, first\ out\ (FIFO)$

   每次檢查當前 $running$ 的 $process$ 是否已經跑完，若是，則指定下個最早 $ready$ 的 $process$ 給 $scheduler()$。

2. $round-robin\ (RR)$

   首先我先建一個 $queue$ 的陣列，並於每次檢查是否有 $process$ 正在跑，沒有則從 $queue$ 中取最前面的給 $scheduler()$，有則檢查當前 $running$ 的 $process$ 是否已經跑完，若是則將此 $process$ pop 出來，若不是則將此 $process$ pop 出來再 $push$ 到 $queue$ 的最後面，接著取最前面的給 $scheduler()$。

3. $shortest\ job\ first\ (SJF)$

   每次先檢查當前 $running$ 的 $process$ 是否已經跑完，若是，則找出執行時間最短且 $ready$ 的 $process$ 並指定給 $scheduler()$。

4. $preemptive\ shortest\ job\ first\ (PSJF)$

   與 SJF 相似，只是少了檢查當前 $running$ 的 $process$ 是否已經跑完的步驟，意即在每個時間點任何已經 $ready$ 都有資格可以搶走 $core$。

在 $new\_proc\_exec()$ 中，我先用 $fork()$ 創建一個 $child\ process$，接著根據是 $child\ process$ 還是 $parent\ process$ 做出相對應的操作。

1. $child\ process$

   呼叫自定義的 $system\ call\ (get\_time)$ 來取得 $child\ process$ 開始的時間，再根據它的執行時間跑相對應數量的 $unit\_time$，接著再呼叫一次 get_time 來取得 $child\ process$ 結束的時間，再呼叫自定義的 $system\ call\ (my\_printk)$ 來寫出 " $[Project1]$ $[pid]$ $[start\ time]$ $[end\ time]$ "。

2. $parent\ process$

   將 $child\ process$ 綁定在一個 $core$ 上 ($core$ 1)。

2. 核心版本

$linux$ $4.14.25$

3. 比較實際結果與理論結果，並解釋造成差異的原因

跑出來的實際結果會比理論結果快。

因為一個 $unit\_of\_time$ 的時間可能會小於 $fork()$ 或 $context\_switch$ 的時間，因此實際結果可能比較快。