Report of Operating System Project 1

戴佑全
黃子賢
劉彥廷

April 26, 2016

1 FIFO

1.1 Design

資料結構：一開始打算以link list來當作ready queue，但後來想到有給process的數量再 加上並沒有用到link list可以在中間隨意插入的優點，所以改用array當作ready queue， 並有2個變數紀錄現在執行的child與下一個要被fork的child。

FIFO是一個很直觀的 scheduler，因爲只有一顆CPU，所以我的作法是parent將 child要跑多少時間跟child說，之後將child的priority提到最高，而 parent 會重新拿到CPU有2種情況，

1. 因爲可能中途要fork其他的child，所以child需要在那時將CPU還給parent，這裡使用的方式是將parent的priority設爲98，換child時將child提成99，當child要回給 parent時將自己設爲97，就能完成context switch了，而因爲此時child還沒執行完，只是暫時換parent，所以在parent中現在執行的變數不變，所以fork完之後會接續做。

2. 當child執行完後，輪到parent時，將parent中現在執行的變數增加並wait child 以防有zombie。 所以parent的流程是，先檢查是否在idle(現在執行的child = 下一個要被 fork的child)，如果是則做for loop直到下一個child被出來，再來是檢查需不需要fork， 因爲並非每次輪到parent時都要fork，再來是將child需要跑的時間告訴child，時間爲 min(下一個要被fork的child的 ready time, 現在執行的child的execution time)。

1.2 Result

1. FIFO_2.txt

```
P1 3515
P2 3516
P3 3517
P4 3518

[Project1] 3515 1461500526.987120466 1461500691.939705154
[Project1] 3516 1461500691.939706083 1461500702.258074897
[Project1] 3517 1461500702.258075945 1461500704.295347989
[Project1] 3518 1461500704.295349110 1461500706.332647694
```

1.3 Comparison

與理想中的差不多，但是在dmesg稍微有點誤差，原因大概是parent 與child之間context switch造成的，除了priority有保護順序外，因爲FIFO並不考慮preemptive，所以只有上一 個做完下一個才會做，再加上每次child死亡都會wait所以這也會保護到答案順序的正確。

2 SJF

2.1 Design

設計跟FIFO差別不大，唯一的差別是在fork完之後要做一次sort，有2種情況，

1. 現在有child正在執行中，那麼只sort除了這支child的其他已被fork出來的child。
2. 現在沒有child正在執行中，那麼sort所有已被fork出來的child。除此之外剩下皆與FIFO相同。

2.2 Result

1. `SJF_3.txt`

```
P1 2730
P4 2733
P5 2734
P6 2735
P7 2736
P2 2731
P3 2732
P8 2737

[Project1] 2730 1461506590.620579045 1461506597.056205686
[Project1] 2733 1461506597.056208329 1461506597.077219594
[Project1] 2734 1461506597.077221749 1461506597.098146308
[Project1] 2735 1461506597.098147463 1461506605.898578152
[Project1] 2736 1461506605.898581016 1461506614.651940234
[Project1] 2731 1461506614.651943364 1461506625.677029525
[Project1] 2732 1461506625.677032127 1461506641.046276916
[Project1] 2737 1461506641.046279188 1461506660.897146737
```

2.3 Comparison

SJF與FIFO很像，都是不preemptive，所以同FIFO，順序並不太會錯誤，在來就是因爲要頻繁的sort所以在時間誤差方面會比較大。

3 PSJF

3.1 Design

1. Data structure: A poor man's priority queue, implemented by an array and qsort. The smaller one's exction time is, the higher its priority is.
   (a) Insert: $O(n \log n)$
   (b) Pull: $O(1)$
2. Main while loop:

```
while(there exists non-finished processes) do
  if(clock is the next ready process's ready time) do
    fork this process
    insert this process to priority queue
  else
    wait until next ready process
    continue
```

```
    end

    pull the process from the priority queue
    if(next ready process's ready time is smaller than this process's finish time) do
      this process can only run (next ready process's ready time - clock)
    else
      this process can run til it finishs
      wait this child, and mark this process as finished
    end
  end
```

3.2 Result

1. PSJF_2.txt

```
P2 3195
P1 3194
P4 3197
P5 3198
P3 3196

[Project1] 3195 1461556744.671319569 1461556746.818243380
[Project1] 3194 1461556742.541598901 1461556751.107887218
[Project1] 3197 1461556753.257726208 1461556757.596873203
[Project1] 3198 1461556757.596918243 1461556759.755637682
[Project1] 3196 1461556751.107896674 1461556766.169973474
```

3.3 Comparison

Basicly, the result is as same as prediction.

4 Contribution of Each Member

戴佑全: system call, RR

黃子賢: child, FIFO, SJF

劉彥廷: main, PSJF