OS project 1

楊子平 b07902136

設計

CPU

- 使用CPU0運行scheduler，並且將此程式的priority調高。
- 使用CPU1運行child process，並且將正在運行的process priority調高，其餘priority調低。

檔案

- **process.h**：定義process結構與UNIT_TIME。
- **main.c**：輸入並呼叫scheduler
- **scheduler.c**：排程

scheduler

- 使用一個迴圈，每次將time加一，模擬每次多一個time unit時會發生的事。每輪迴圈會依序跑以下的行為。
  1. 如果正在跑的程式該跑完了，wait它。
  2. 如果有此時開始ready的process，fork出一個子程序並執行它(並且將priority調低)。
  3. 選擇這個時間點應該要執行的子程序(select run函數)，將之priority調高，將正在執行的子程序priority調低。
  4. 跑一個Unit_Time。

select run

- **FIFO**：在一開始就按照ready time來排序，因此直接選擇下一個尚未執行完的程序即可。

- **RR**：維護一個queue，每次都執行queue最前端的process，執行結束之後就將之pop掉。當執行過time quantum的倍數後，將之pop掉改放在最後面。新進的process也放在最後面。**如果同時有跑超過time quantum的process與新進的process，新進的process放在前面。**
- **SJF**：如果已經有正在執行的程序，繼續執行它，如果沒有的話，O(n)找尋時間最小的程序。
- **PSJF**：O(n)找尋剩餘時間最小的程序。

其他函數實作細節

- **set_priority**：使用**sched_setscheduler**，如果要設為高優先度，用參數**SCHED_OTHER**，並將nice值調為-19。如果要設為低優先度，用參數**SCHED_IDLE**。
- **assin_cpu**：使用**sched_setaffinity**函數實作。
- **syscall(334, ...)**：取得時間的syscall，使用**getnstimeofday**實作。
- **syscall(335, ...)**：輸出到dmesg的**syscall**，使用**printk**實作。

執行方式

```
make
sudo ./a.out
```

核心版本

- **kernel**：Linux 4.14.25 x86_64
- **System**：Ubuntu 16.04.6 LTS

比較

根據不同的scheduling 方式，結果呈現不同的誤差。

- **FIFO**：FIFO的誤差有正有負，但大致都在5%以內，我認為是因為**TIME_MEASUREMENT**就是FIFO，因此這部分的誤差還算小。
- **RR**：RR的誤差比較多負的(實際執行時間比較短)，但是誤差多在2~3%之間，也算是誤差比較小的scheduling。

- **PSJF**： PSJF的誤差多是負的(實際執行時間比較短)，且誤差較小(約1~2%)
- **SJF** : SJF的誤差多是正的(實際執行時間比較長)，誤差量也比前三者大，我認為是因為SJF需要O(n)的時間去找到下一個執行的項目，因此較花時間。

總和平均來說，實際執行時間要比理論執行要長一些，我認為是因為，TIME_MEASUREMENT 是用FIFO去測量，由於FIFO的排程方式不需要太多的運算，因此FIFO算出來的time unit可能比實際上的短一些，然後用此來衡量其他測資，就會使其他測資看起來跑太慢。

另外，即使同為FIFO的測資，仍然有約5%的誤差，因此這個範圍可能為正常不同測資間的浮動，這些浮動也可能因為不同的執行情形也有所改變。我認為可能的原因之一是，在fork出去之後，如果先執行child process，那麼就會沒有將之設定為低優先序，使其先運行。這個問題在不同次執行之間也會發生，是屬於在user space實作就難以避免的問題。

Reference

B07902075 林楷恩
B07902123 蔡奇夆
B07902131 陳冠宇
B07902134 黃于軒
B07902141 林庭風
B07902028 林鶴哲