作業系統 Project 1 報告

B07902129 何政勳

一、 設計

1. Job

每個工作會記錄名字、到達時間、剩餘時間、是否到達以及子程序的 pid。
到達時間用於判斷哪個工作先來，哪個工作後來。
剩餘時間用於紀錄一份工作還需要多少個 unit time。
是否抵達用於排程時進行判斷是否要考慮這個工作。
子程序的 pid 讓母程序可以掌握子程序的狀態。
以上資料儲存於一個 Job structure:

```
typedef struct job {
    char name[32];
    int ready;
    int remain;
    int isReady;
    pid_t pid;
}
```

2. Scheduler

scheduler 跑在一個 while loop 中，直到所有工作都完成為止。
每個 loop 開始時，scheduler 會檢查是否有工作完成了，以及是否所有工作都完成了，若是則會跳出迴圈。
scheduler 接著會檢查是否有工作已經 ready，若有會將該工作的 isReady 設成 1。
再來 scheduler 會決定下個 unit time 該由哪個工作執行，若該工作的程序還沒被 fork 則會被 fork 出來，並提高其優先度。
最後 scheduler 跑一個 unit time，時間則會累加一，而正在跑的工作的 remain 減一。

以下說明不同 policy 選擇的方法

(1) FIFO

若當前有正在執行的工作，則繼續執行。若無，則從已經抵達(isReady == 1)的所有程序中選擇最早抵達(最小 ready)的執行。

(2) SJF

若當前有正在執行的工作，則繼續執行。若無，則從已經抵達(isReady == 1)的所有程序中選擇最少工作量(最小 remain)的執行。

(3) PSJF

從已經抵達(isReady == 1)的所有程序中選擇最少工作量(最小 remain)的執行。

(4) RR

若當前有正在執行的工作，則檢查其運行時間是否為 500 的倍數，若不是則繼續執行。其他則從已經抵達(isReady == 1)的所有程序中順序執行下一個正在等待的工作。

3. 實作

實作時 scheduler 與 child processes 分別跑在不同的 CPU，以確保 scheduler 不會被 child processes 阻擋或 preempt。

當一個 child process 被 fork 出來之後，scheduler 會用 sched_setaffinity 將其指定到另一個 CPU，並用 sched_setscheduler 將其排程策略設定成 SCHED_IDLE，讓他被真的 CPU 選到的機會降低。

當一個 child process 被選到之後，scheduler 會用 sched_setscheduler 將其排程策略設定程 SCHED_OTHER，讓他能正常的被 CPU 選到。

## 二、 環境

這次作業的 kernel version 是 5.6.3

作業系統為 elementary OS

CPU 為 intel i7-8700

## 三、 差異

以 FIFO_1 為例，輸出為

[ 1025.468065] [Project1] 6029 1024.882050429 1025.486887488

[ 1026.064438] [Project1] 6030 1025.487060513 1026.83285147

[ 1026.666973] [Project1] 6038 1026.83468296 1026.685845458

[ 1027.256855] [Project1] 6039 1026.686024898 1027.275755949

[ 1027.829805] [Project1] 6040 1027.275896595 1027.848730486

即使輸入的時間都為 500 單位，五個子程序的起始與結束時間還是有所相差。

除了程序 run time 本身就會有差別外，由於 scheduler 需要等 child process fork 出來之後才能降低其優先度，但無法保證在 fork 完後是由 parent 或是 child 先跑。因此 child 有可能會先偷跑一些，也有可能造成過早計時。

此外雖然 child process 的 scheduling policy 被設定程 SCHED_IDLE，這僅代表其被 CPU 選到的機會很少，但仍然有機會被 CPU 選到，造成一些偷跑的可能。

本次作業中使用的是 clock_gettime()以及 CLOCK_MONOTONIC。在工作時長很短時(例如 500 單位)，有時取得的結束時間會比開始時間還早一些，通常是 tv_sec 一樣，而 tv_nsec 卻比較早，目前尚未找到確切原因。