Report

b07902076 資工二 許世儒

1 設計

(1) 首先，我先寫了兩個system call分別是get_time.c以及print_time.c
get_time.c： 利用getnstimeofday() 來回傳一個long long int代表當前時間(ns)
print_time.c: 把給定的時間(ns)轉換成秒再用printk印出去

(2) 在process.c中，主要有幾個function提供main來使用，其中包括

(a) unit_time(): 用來當作一個單位時間

(b) set_priority(): 用來改變process在cpu中的priority

(c) assign_cpu(): 用來指定process到某個cpu上(在這個project中，我使用2個core，core編號0用來跑parent process，編號1用來跑child process）

(3) 在main.c中，主要分成4個部分對應到4個schedule algorithm

(a) FIFO: 首先進到scheduler_FIFO()中，會先空轉直到timer到達第一個process的ready time，之後再用while loop去檢查timer是否大於所有的process的ready time，如果有， 就call assign_FIFO()而assign_FIFO()會先create一個child process並把他指定在第1號cpu上同時把他的priority設成較高的值，他就會開始執行 ，而parent process就會等到他執行完wait()，並且將timer加上該process執行的時間繼續while loop直到所有process皆完成。

(b) RR: scheduler_RR大致和scheduler_FIFO相同，唯一不同之處在於，每assign一個process時，在RR中不會馬上wait()，而在assign_RR()也會先fork出一個child process，並將child process指定到1號cpu，並把他的priority提高，再跑兩層for loop第1層是該process的exec_time，第2層是每500秒會讓出cpu讓其他人跑，如果exec_time小於500他就會直接跑完然後再找下一個process執行，如果他大於500就會在跑到500時讓出cpu(此處我使用sched_yield()來實作)，在找下一個已經assigned的process，最後直到所有process皆執行完成。

(c) SJF: 此處會用到我自訂的struct中的status變數，主要是紀錄該process目前的狀態，0代表尚未被assign，1代表已經ready，2代表已經執行完成。並且我寫了一個select_task()是從所有process找一個已經ready但是尚未完成並且是所有剩餘process中執行時間最短的process。首先，我會先找到一個已經ready且執行時間最短的process，如果找不到的話就用一個while loop直到找到為止，再將已經找到的process丟進1號cpu該使執行，執行完後parent process再去wait()並將他的status設為2，重複直到所有process皆完成。

(d) PSJF: 在PSJF中，會有兩種情況導致正在執行的process被中斷，其一為有新的process已經ready且該process的執行時間短於當前正在執行的process剩餘執行時間，或者是當前process已經執行完畢，要再找下一個process去執行，因此，我會在每輪都知道該process是否會執行完，或者是讓出cpu，首先，如果是最後一個process在執行的話，該process一定會執行完，如果是一般情況，我們可以去比較到下一個process的ready_time與當前process的剩餘執行時間，來判斷正在執行的process會不會執行完並將這個時間差紀錄下來(running_time)，並執行running個unit_time()之

後我們就可以判斷該process是否已經完成(exec_time是否為0)如果尚未為0我們就把該process的priority降低，之後再用while loop去找可以assign的process並將他們丟進1號cpu(丟進去同時降低priority)，最後再用select_task()找可以執行且執行時間最短的process並提高priority，重複直到所有process皆完成。

2 核心版本

Kernel Version: Linux 4.14.25
Platform: VMWare15
OS: Linux Ubuntu 16.04

3 比較

```
File output/FIFO_1_stdout.txt : optimal is 10.474427522 seconds, real is 10.206122680 seconds, ratio = 0.974384773
File output/FIFO_2_stdout.txt : optimal is 471.209579435 seconds, real is 492.304258624 seconds, ratio = 1.044767085
File output/FIFO_3_stdout.txt : optimal is 155.719822487 seconds, real is 149.438672659 seconds, ratio = 0.959663775
File output/FIFO_4_stdout.txt : optimal is 10.614086555 seconds, real is 11.027870095 seconds, ratio = 1.038984376
File output/FIFO_5_stdout.txt : optimal is 153.066300848 seconds, real is 159.253012047 seconds, ratio = 1.040418506
File output/RR_1_stdout.txt : optimal is 10.474427522 seconds, real is 10.538721348 seconds, ratio = 1.006138171
File output/RR_2_stdout.txt : optimal is 22.764422480 seconds, real is 22.874662426 seconds, ratio = 1.004842642
File output/RR_3_stdout.txt : optimal is 174.154814924 seconds, real is 171.724638175 seconds, ratio = 0.986045883
File output/RR_4_stdout.txt : optimal is 127.788015763 seconds, real is 126.046001674 seconds, ratio = 0.986367939
File output/RR_5_stdout.txt : optimal is 127.927674796 seconds, real is 130.463460217 seconds, ratio = 1.019822024
File output/SJF_1_stdout.txt : optimal is 35.892371640 seconds, real is 35.348296704 seconds, ratio = 0.984841488
File output/SJF_2_stdout.txt : optimal is 39.104529414 seconds, real is 39.016462163 seconds, ratio = 0.997747902
File output/SJF_3_stdout.txt : optimal is 131.363287023 seconds, real is 133.893986535 seconds, ratio = 1.019264892
File output/SJF_4_stdout.txt : optimal is 27.931806724 seconds, real is 27.145899047 seconds, ratio = 0.971863343
File output/SJF_5_stdout.txt : optimal is 11.172722690 seconds, real is 10.801821748 seconds, ratio = 0.966802994
File output/PSJF_1_stdout.txt : optimal is 71.226107146 seconds, real is 74.769012178 seconds, ratio = 1.049741663
File output/PSJF_2_stdout.txt : optimal is 23.742035715 seconds, real is 25.809113840 seconds, ratio = 1.087064064
File output/PSJF_3_stdout.txt : optimal is 6.982951681 seconds, real is 7.579635083 seconds, ratio = 1.085448594
File output/PSJF_4_stdout.txt : optimal is 34.635440338 seconds, real is 38.395299951 seconds, ratio = 1.108555271
File output/PSJF_5_stdout.txt : optimal is 39.104529414 seconds, real is 41.496001167 seconds, ratio = 1.061155876
```

這是我用程式執行出來的結果，$ratio = \frac{real}{opt}$
我發現大多數的實際時間會多餘理論時間，我認為主要的原因是因為實際上scheduler不只要紀錄時間，還要排程並且紀錄各個process的狀態，這些都是可能造成實際時間多餘理論時間的原因。
而有些實際時間少於理論時間，我認為可能的原因是因為fork出新的child process及實際調整他的priority之間可能會有時間差，而這個時間差就可能讓child process偷偷使用CPU，因而造成這樣的結果。