HW07 CPU 與 I/O 的平行化

繳交說明：

1. 用來量測的程式

$ sudo apt install sysbench

$ sysbench --test=fileio --num-threads=20 --file-total-size=1G --file-test-mode=rndrw prepare

#更改參數必須要用 sudo sysctl -w kernel.sched_wakeup_granularity_ns=0

不能用 echo

2. 執行結果截圖

(1)第一次實驗 100 個 threads ,write_size 10G

$ sysbench --test=fileio --num-threads=100 --file-total-size=10G --file-test-mode=rndrw prepare

項目	使用預設參數 (kernel.sched_wakeup_granularity_ns =10000000)	更改參數 (kernel.sched_wakeup_granularity_ns =0)
User	0.10 s	0.09s
System	10.78 s	10.09s
Cpu	108%	100%
Total	10.067	10.080

結論：可能因為是 SSD 寫很快，於是想說增加 write_size 10G ->30G

(2)第二次實驗 100 個 threads ,write_size 30G

$sysbench --test=fileio --num-threads=100 --file-total-size=30G --file-test-mode=rndrw prepare

項目	使用預設參數 (kernel.sched_wakeup_granularity_ns =10000000)	更改參數 (kernel.sched_wakeup_granularity_ns =0)
User	0.10 s	0.10 s
System	10.45 s	10.68 s
Cpu	104%	107%
Total	10.049	10.062

結論：發現仍然差異不大

3. iostat 量測的截圖(因兩次實驗差不多，故第一次實驗作為代表)

(1) kernel.sched_wakeup_granularity_ns =10000000

Device	tps	kB_read/s	kB_wrtn/s	kB_read	kB_wrtn
loop0	0.01	0.02	0.00	112	0
loop1	0.00	0.01	0.00	47	0
loop2	0.01	0.02	0.00	128	0
loop3	0.02	0.18	0.00	1150	0
loop4	0.01	0.05	0.00	328	0
loop5	0.05	0.21	0.00	1308	0
loop6	0.02	0.18	0.00	1105	0
loop7	0.00	0.01	0.00	47	0
sda	33.88	308.33	4076.68	1940718	25659917

(2) kernel.sched_wakeup_granularity_ns =0

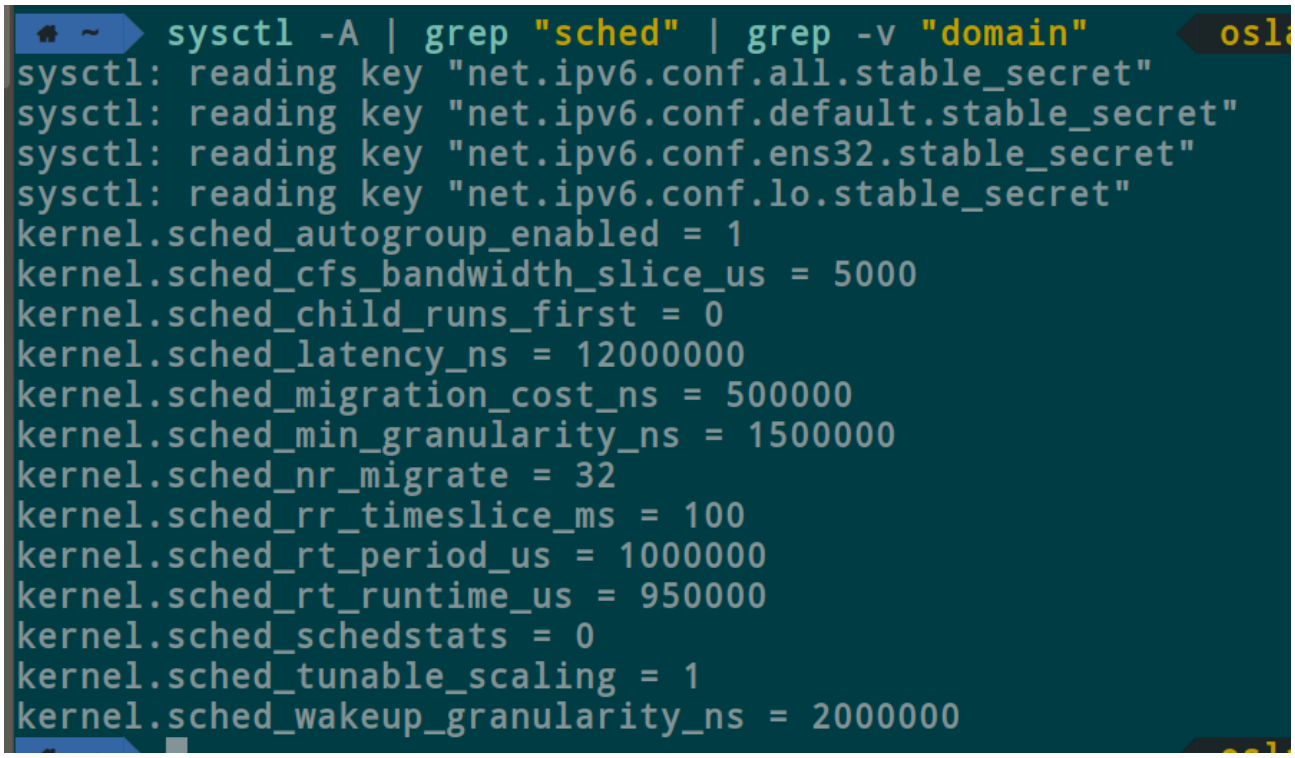
Device	tps	kB_read/s	kB_wrtn/s	kB_read	kB_wrtn
loop0	0.01	0.02	0.00	112	0
loop1	0.00	0.01	0.00	47	0
loop2	0.01	0.02	0.00	128	0
loop3	0.02	0.19	0.00	1150	0
loop4	0.01	0.05	0.00	328	0
loop5	0.05	0.21	0.00	1308	0
loop6	0.02	0.18	0.00	1105	0
loop7	0.00	0.01	0.00	47	0
sda	32.17	300.75	4116.61	1853986	25376925

4. iostat 欄位說明

欄位	說明
Device	Device name
Tps	每秒鐘進行多少個 I/O 指令
KB_read/s	每秒鐘讀多少次
KB_wrtn/s	每秒鐘寫多少次
KB_read	從開機以後讀的數量
KB_wrtn	從開機以後寫的數量

//youtube 筆記

1. sysctl -A | grep "sched" | grep -v "domain “ #
   #kernel.sched_latency_ns #Kernel 拿到 CPU 可以執行多久 Response time expected value
   #kernel.sched_migration_cost_ns #CPU 計算假設從一個核心換到另一個核心時間
   #kernel.sched_min_granularity_ns #最小拿到的 timeslice 數量
   #rt 為 real time
   #若負載很大 超過 exceeds sched_latency_ns/sched_min_granularity_ns = 12000000/50000= 24 個 TASK，會造成回應時間變慢
   #timeslice 變成 number_of_running_tasks * sched_min_granularity_ns
   # kernel.sched_wakeup_granularity_ns # vruntime



2. 下載 Kernel
   wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.4.2.tar.xz

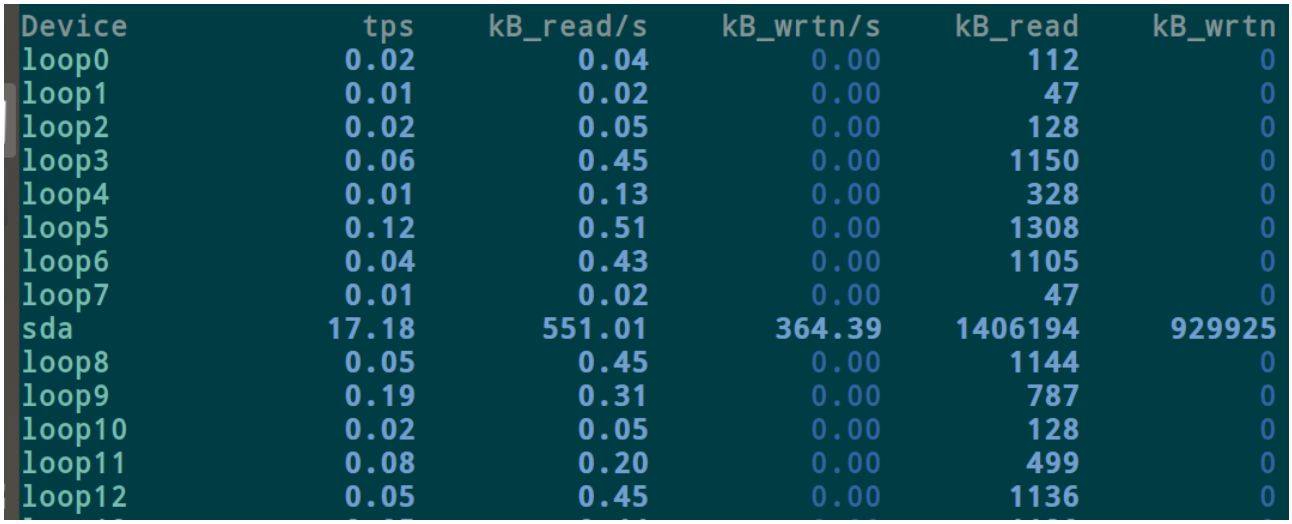
3. 安裝
   sudo apt-get install git fakeroot build-essential ncurses-dev xz-utils kernel-package
4. 編譯核心
   make meunconfig

5. 比較(待補)

make clean

time make -j8

iostat 看 rw 效率 # watch "iostat | grep sda" 可用這樣持續看



Device	tps	kB_read/s	kB_wrtn/s	kB_read	kB_wrtn
loop0	0.02	0.04	0.00	112	0
loop1	0.01	0.02	0.00	47	0
loop2	0.02	0.05	0.00	128	0
loop3	0.06	0.45	0.00	1150	0
loop4	0.01	0.13	0.00	328	0
loop5	0.12	0.51	0.00	1308	0
loop6	0.04	0.43	0.00	1105	0
loop7	0.01	0.02	0.00	47	0
sda	17.18	551.01	364.39	1406194	929925
loop8	0.05	0.45	0.00	1144	0
loop9	0.19	0.31	0.00	787	0
loop10	0.02	0.05	0.00	128	0
loop11	0.08	0.20	0.00	499	0
loop12	0.05	0.45	0.00	1136	0

6. 核心程式碼

#等於 1 的時候重新排程

#*cur 為目前 TASK *se 代表是完成 IO TASK

https://elixir.bootlin.com/linux/latest/source/kernel/sched/fair.c#L6941

```
1.  if (wakeup_preempt_entity(se, pse) == 1) {
2.      /*
3.       * Bias pick_next to pick the sched entity that is
4.       * triggering this preemption.
5.       */
6.      if (!next_buddy_marked)
7.          set_next_buddy(pse);
8.          goto preempt;
9.  }
10. return;
11. preempt:
12. resched_curr(rq);
```

https://elixir.bootlin.com/linux/latest/source/kernel/sched/fair.c#L6894

```
1.  static int wakeup_preempt_entity(struct sched_entity *curr,
2.              struct sched_entity *se) {
3.      s64 gran, vdiff = curr->vruntime - se->vruntime;
4.      if (vdiff <= 0)
5.          return -1;
6.      gran = wakeup_gran(se);
7.      if (vdiff > gran)
8.          return 1;
9.      return 0;
10. }
```

7. 安裝 I/O 比較程式 sysbenck

可參考 https://blog.toright.com/posts/5051/linux-disk-io-%E6%95%88%E8%83%BD%E6%B8%AC%E8%A9%A6.html