HW2

4.2

需要增加concurrency的時候，multithreading 表現會比single-threading還好

4.4

作業系統並不會分配不同process的thread在不同的processors上，因此，single-processor system跟multiprocessor system 在效能表現上並無太大差異

4.13

(a) kernel thread 比處理器少的時候，某些處理器會閒置

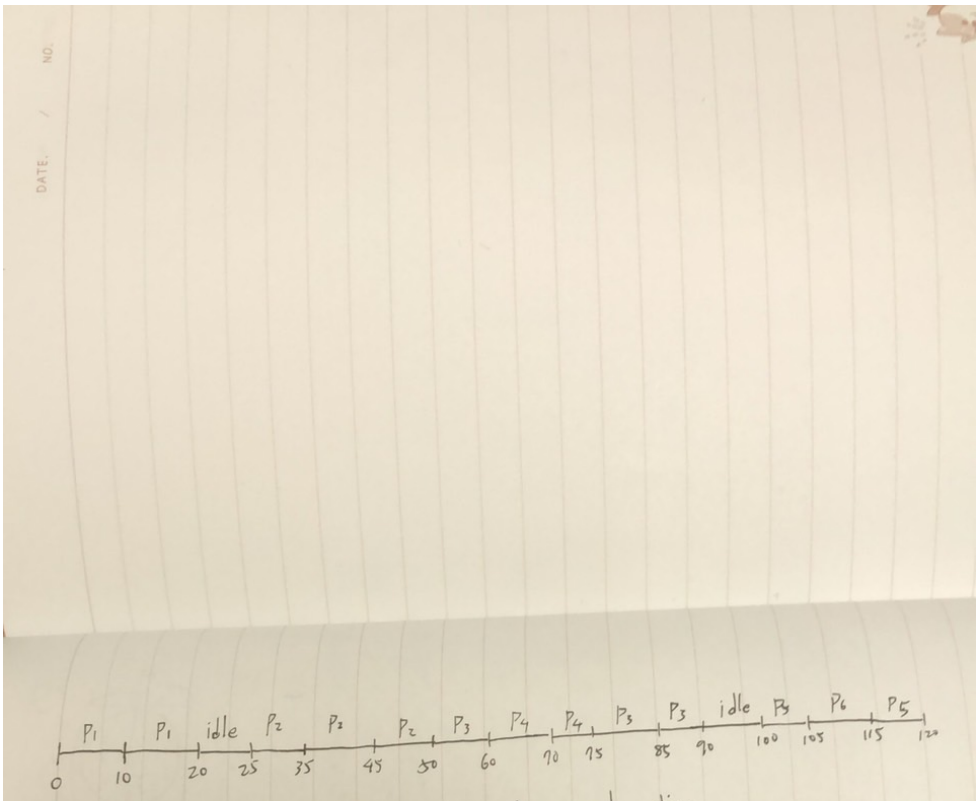
(b) kernel thread 跟處理器一樣的時候，有可能所有的處理器都能被配給thread

(c) kernel thread 比處理器多的時候，能透過swap讓其他準備好的kernel thread來執行，藉此增加效能

5.6

這種排序法較適合使用在CPU bound上 ，因為當他消耗完整個quantum後，就會獲得更高的優先權

5.8



waiting time

P_1 0

P_2 0

P_3 $(50-30)+(75-60)=35$

P_4 0

P_5 $(115-105)=10$

P_6 0

turnaround time

P_1 $0+20=20$

P_2 $0+25=25$

P_3 $35+25=60$

P_4 $0+60=60$

P_5 $10+10=20$

P_6 $0+10=10$

5.10

FCFS : no starvation, 因為每個最後都有機會

Shortest job first : 較長的處理會使得有更多的等待時間，就有可能導致starvation

Round robin : no starvation, 每個在經過quantum後，都會有機會

Priority : 優先度過低的可能會starvation

5.15

FCFS : 是以時間先後順序來排的，因此如果short job時間排在long job後，就會使得等待時間很久

Round robin : 因為每個job都是執行quantum因此short job很快就可以被輪到，並執行完成

Multilevel feedback queue：跟RR類似可以對short job執行有利的調整

6.4

如果一個user-level program 可以去disable interrupts，那它就能去阻止context switching的發生

6.10

與semaphore類似的方式,每個mutex lock都會有一個等待佇列，當呼叫的時候mutex lock無法使用時就會被放入等待佇列，而當lock被release時，就從佇列挑出process並喚醒process

6.11

當lock短的時候，使用spin lock會比較好，若使用mutex會產生額外的的overhead

當lock長的時候，使用mutex會比較好，若使用spin lock會產生busy waiting的問題

使用mutex會比起spin lock所造成的busy waiting有更少的效能浪費