4.1

多線程適合高密集計算與大量 IO 操作，因此以下兩種情形並不會比單線程較率還高：

1. 高密集計算、少量 IO 操作：在計算上相比單線程，需要額外的線程管理與增加上下文切換與同步開銷。
2. 低密集計算、大量 IO 操作：這時跟 CPU 性能較無關，而是 IO 操作速度，而單線程能更好發揮 IO 操作的異步性能。

4.3

所有線程都共享：Heap memory, Global variables
各線程單獨持有：Register values, Stack memory

4.4

在單一處理器上運行多線程會需要將線程作時間切割來共享 CPU 資源，增加了上下文切換與線程調度成本，降低了效率。
在多處理器上運行多線程能讓不同線程在不同 CPU 上同時執行，節省上下文切換與線程之間的等待，有比單一處理器更好的效能表現。

5.2(a)

CPU 利用率越高代表可以將更多進程調度到 CPU 上執行，一個進程長時間使用 CPU 時，會讓其他進程的響應時間便久。

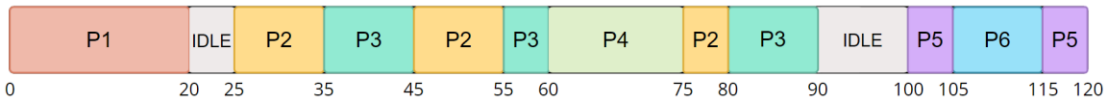
5.2 (b)

透過優先調度短進程讓平均周轉時間縮短，但可能讓長周轉時間的進程等待時間變長。

5.2 (c)

如果 I/O 使用率高，進程在等待 I/O 操作完成前，CPU 不能使用 I/O 設備，CPU 利用率降低；CPU 利用率高則 I/O 使用率降低，I/O 較常處於閒置狀態，會導致進程要等 I/O 操作完成才能繼續執行，造成更久的等待時間。

5.8(a)



5.8(b)

各進程周轉時間

P1 = 20 – 0 = 20

P2 = 80 – 25 = 55

P3 = 90 – 30 = 60

P4 = 75 – 60 = 15

P5 = 120 – 100 = 20

P6 = 115 – 105 = 10

5.8(c)

各進程等待時間

P1: 20 – 20 = 0

P2: 55 – 25 = 30

P3: 60 – 25 = 35

P4: 15 – 15 = 0

P5: 20 – 10 = 10

P6: 10 – 10 = 0

5.8(d)

CPU 利用率 = (105 / 120) * 100% = 87.5%

5.14(a)

進程在被執行時優先度增加的速率更快，所以會將進行中的進程完全執行完後才會換下一個進程執行，稱為 FCFS。

5.14(b)

進程在被執行時優先度減少的速率更快，所以每當有新進程抵達都會優先執行新進程，又稱為 LIFO。

5.15(a)

FCFS 為非搶占式的排程算法，並不會對短進程有更好的執行效率，每個進程都有相對應的等待時間和執行時間。

5.15(b)

RR 透過時間切片，來讓同樣在等待的進程能依照切片來輪流執行，這對於短進程有更好的表現，能在較短的時間切片中迅速完成，提高短進程的執行效率。

5.15(c)

Multilevel feedback queues 是一種複雜的排程算法，根據進程的屬性分配到不同的隊列中，每個隊列有不同的優先度和時間片大小。短進程會被分配到優先度較高的隊列中，進程等待的時間會因為優先級和時間片大小的不同而有所差異。因此，短進程等待時間較短。

6.4

用戶級的程序沒有完全控制中斷的權限，可能會導致程序無響應，直到同步原語完成並啟用中斷才會繼續執行，這會導致進程或線程阻塞，影響系統性能與利用率。

6.6

當一個進程嘗試獲取自旋鎖時，不能持有自旋鎖，是為了避免發生死鎖，如果持有自旋鎖的進程試圖獲取一個信號量，而該信號量又被另一個正在等待該自旋鎖的進程所持有，那麼這兩個進程將會相互等待，導致死鎖。

6.10

定義一個等待佇列 Wq，用於存儲等待獲取 mutex lock 的進程。
當一個進程嘗試獲取一個被佔用的 mutex lock 時，將其放入 Wq 中。
當 mutex lock 釋放時，檢查 Wq 中是否還有進程等待，如果有，從 Wq 中取出一個進程，然後將 mutex lock 分配給該進程。