Operating System HW2

1. (10%) Consider a computer that does not have a TEST AND SET LOCK instruction but does have an instruction to swap the contents of a register and a memory word in a single indivisible action. Can that be used to write a routine enter region such as the one found in Fig. 2–12.

Solution:
可以，我們先把 1 存入一個 register，接著再把這個 register 與 LOCK swap，接著判斷現在 register 的值是否為 `0` ，如果為 `0`，代表這個鎖是打開的，可以進去這個 critical region，因為有swap，所以 lock 的值會變成 1，把這個鎖鎖上，如果register的值為 1，代表鎖是關的，重新回到 ENTER_REGION 這個標籤的行數，繼續check lock 的值是否為 `0`

pseudo code:

```
ENTER_REGION:
        mov reg, 1
        swap reg, lock
        bne reg, 0, ENTER_REGION
        RET
```

---

2. (20%) Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for the CPU efficiency (i.e., the useful CPU time divided by the total CPU time) for each of the following: (a) Q = ∞ (b) Q > T (c) S < Q < T (d) Q = S (e) Q nearly `0`

Solution:
(a) $Q = \infty$
Process 不會被中途中斷，每個 process 會有一次在 loading 時的 context switch，因此效率為$\frac{T}{T+S}$
(b) $Q > T$
process 平均只會有一次 context switch，效率為$\frac{T}{T+S}$
(c) $S < Q < T$
每個 process 會有$\lceil \frac{T}{Q} \rceil$次的context switch，效率為$\frac{T}{T+S*\lceil \frac{T}{Q} \rceil}$
(d) $Q = S$
- 若$Q > T$，效率與(a)相同，為$\frac{T}{T+S}$

- 若$Q < T$，將(c)的效率代入，效率為$\frac{T}{T+S*\lceil\frac{T}{S}\rceil} = \frac{T}{T+T} = \frac{1}{2}$

(e) $Q\ nearly\ 0$
將(c)的效率代入，效率為 $\frac{T}{T+S*\lceil\frac{T}{0}\rceil} = \frac{T}{\infty} = 0$，也可以由觀念解釋，因為 process 幾乎沒有執行就會直接進行 context switch，所以效率為 0

---

3. (10%) Consider the interprocess-communication scheme where mailboxes are used. Suppose a process P wants to wait for two messages, one from mailbox A and one from mailbox B. What sequence of send and receive should it execute so that the messages can be received in any order?

Solution:
創建兩個 thread，一個負責接收mailbox A的訊息，另一個接收mailbox B的訊息，而 send 操作並不會影響訊息的接收

Using pthread:

```
pthread_t mailboxAreceiver,mailboxBreceiver;

pthread_attr_t attr;
pthread_attr_init(&attr);

pthread_create(&mailboxAreceiver,&attr,funcOfReceive,&messageOfmailboxA);
pthread_create(&mailboxBreceiver,&attr,funcOfReceive,&messageOfmailboxB);

pthread_join(mailboxAreceiver,NULL);
pthread_join(mailboxBreceiver,NULL);
```

---

4. (10%) Consider the following program that uses the Pthreads API. What would be the output of the program? (Note that the line numbers are for references only.)

Solution:
Output:

A = 1
B = 1
C = 2
D = 2

A, B, C, D 的值是可以確定的，但順序要看作業系統的 scheduler 是怎麼排程的，可以任意排列。

在程式第 13 行進行了第一次的fork，parent process 對應到 A 的輸出，value沒有改動，所以$A = 1$，fork 出來的 child 在第18行又進行了一次 fork ，第二次 fork 出來的 parent 對應到 B 的輸出，這個 process 並沒有對 value 進行改動，輸出為$B = 1$，第二次的 fork 的child 在第 23 行 又進行了一次 fork ，parent 及 child 分別對應到了 C 和 D 的輸出，這兩個 process 分別都創建了一個 thread 來執行 runner，把各自 process 內的 value 都加上 1 ，因為有 pthread_join，所以會等待 thread 運行結束後才執行後面的輸出，而 parent 會輸出 $C = 2$，child 則是$D = 2$

我有讓這段程式碼在我的電腦上執行，發現 A, B 的順序較為固定，分別在前面的第一、二行，但是 C, D 的位置可能會互相交換，以下稍微分析一下這件事發生的原因。

每個 process 輸出之前所呼叫的函式大約如下:
A: 一次 fork()
B: 兩次 fork()
C: 三次 fork() 加上一次 runner()
D: 三次 fork() 加上一次 runner()

因為 A 只有進行過一次 fork()，很高的機會會較早執行到第15行的輸出，因此很可能會先被輸出，而 B 有兩次 fork()，會稍微比 A 晚一些，C、D 呼叫函式的內容差不多，因此較有可能互相競爭，較難預測誰先輸出。

雖然可以大致猜出誰會先輸出，但還是要根據系統 scheduler 的排程，仍沒辦法保證 A, B, C, D 的輸出必定會先被執行。