Ch05，番外篇一 synchronization

1. 找出對的那一種

(1)P0 錯誤

```
trival_atomic_bank.c
    signal(SIGALRM, sigAlarm);
    alarm(1);
    printf("銀行開門做生意讓客戶轉帳\n");
    printAccountSummary();

    //產生執行緒
    pthread_t* tid = (pthread_t*)malloc(sizeof(pthread_t) * numCPU);
    pthread_create(&tid[0],NULL,(void *) moneyTransferP0, (void*)0);
    pthread_create(&tid[1],NULL,(void *) moneyTransferP0, (void*)1);
    //等待執行緒完成
    pthread_join(tid[0],NULL);
    pthread_join(tid[1],NULL);
    //列印結果
    printf("下午三點，銀行關門，進行結帳\n");
```

```
所有的人的錢共有：800000
0行員上線
1行員上線
時間：10點
時間：11點
時間：12點
時間：13點
時間：14點
時間：15點
2883行員收工，共處理51993486次轉帳
2882行員收工，共處理53329341次轉帳
下午三點，銀行關門，進行結帳
客戶0有 1032207812 元
客戶1有 465117799 元
客戶2有 1032783774 元
客戶3有 1111319552 元
客戶4有 955744010 元
客戶5有 456682652 元
客戶6有 -88764127 元
客戶7有 867258491 元
所有的人的錢共有 5832349963
共進行105.323 百萬次轉帳
```

(2)P1 正確

```
trival_atomic_bank.c
    alarm(1);
    printf("銀行開門做生意讓客戶轉帳\n");
    printAccountSummary();

    //產生執行緒
    pthread_t* tid = (pthread_t*)malloc(sizeof(pthread_t) * numCPU);
    pthread_create(&tid[0],NULL,(void *) moneyTransferP1, (void*)0);
    pthread_create(&tid[1],NULL,(void *) moneyTransferP1, (void*)1);
    //等待執行緒完成
    pthread_join(tid[0],NULL);
    pthread_join(tid[1],NULL);
    //列印結果
    printf("下午三點，銀行關門，進行結帳\n");
    printAccountSummary();
    long kpi_sum=0;
    for(i=0; i<numCPU; i++)
        kpi_sum+=kpi[i];
    printf(YELLOW"共進行%.3f 百萬次轉帳\n"RESET, ((double)kpi_sum)/1000000);
}
```

```
所有的人的錢共有：800000
0行員上線
1行員上線
時間：10點
時間：11點
時間：12點
時間：13點
時間：14點
時間：15點
3006行員收工，共處理63267490次轉帳
3007行員收工，共處理59796957次轉帳
下午三點，銀行關門，進行結帳
客戶0有 167176640 元
客戶1有 -218958916 元
客戶2有 313833901 元
客戶3有 501313881 元
客戶4有 393864445 元
客戶5有 -285166935 元
客戶6有 -989912504 元
客戶7有 118649488 元
所有的人的錢共有：800000
共進行123.064 百萬次轉帳
```

2. 然後簡單「說明」一下為什麼是對的

(1)因為 P1 使用了 atomic_fetch_add 和 atomic_fetch_sub 可以保證不會互相競爭去改寫

(2)而 P0 使用了以下程式並且平行執行：

假設有兩個 P0 程式同時執行(為了方便辨識，修改了變數名稱，但程式碼架構不變)

行員一

```
long P1-tmp1 = bankAccount[source]-amount1;
long P1-tmp2 = bankAccount[dest1]+amount1;
atomic_store(&bankAccount[source], P1-tmp1);
atomic_store(&bankAccount[dest1], P1-tmp2);
```

行員二

```
long P2-tmp1 = bankAccount[source]-amount2;
long P2-tmp2 = bankAccount[dest2]+amount2;
atomic_store(&bankAccount[source], P2-tmp1);
atomic_store(&bankAccount[dest2], P2-tmp2);
```

(i)設兩支平行 P0 程式 source 剛好相同，

bankAccount[source] = 1000 ，bankAccount[dest1] = 3000，bankAccount[dest2] = 5000

amount1 = 500，amount2 = 1000

並且有可能用以下順序執行：

```
long P1-tmp1 = bankAccount[source]-amount1;  // 500 = 1000-500
long P2-tmp1 = bankAccount[source]-amount2;  // 0 = 1000-1000
long P2-tmp2 = bankAccount[dest2]+amount2;    // 6000 = 5000+1000
atomic_store(&bankAccount[source], P2-tmp1); //   bankAccount[source]  = 0
long P1-tmp2 = bankAccount[dest1]+amount1;    // 3500 = 3000 +500
atomic_store(&bankAccount[source], P1-tmp1); // bankAccount[source]  = 500
atomic_store(&bankAccount[dest2], P2-tmp2);   // bankAccount[dest2] = 6000
atomic_store(&bankAccount[dest1], P1-tmp2);   // bankAccount[dest1] = 3500
```

故原本的 bankAccount[source] + bankAccount[dest1]+ bankAccount[dest2]

= 1000+ 3000 +5000 = 9000

交易完後造成 bankAccount[source] + bankAccount[dest1]+ bankAccount[dest2]

= 500 + 6000 +3500 = 10000

總金額不一致 故錯誤