國立雲林科技大學

作業系統

# Synchronization 與 Deadlocks

B10713138 蔡尚哲

目錄

## 一、同步應用

在某個研究區，在進行實驗，實驗室最多只有三間，因為環境因素，必須同時進行實驗，且要在一定時間內完成，否則將會失準，所以他們分三組實驗室(thread)進行，他們把人劃分成失眠(下面以 a 代稱)，偶爾失眠(下面以 b 代稱)和無失眠(下面以 c 代稱)三種類型，使用穿戴裝置進行實驗，因為三間實驗室的穿戴裝置多寡不同，而且不同時間系統的負載量會不同，所以每次實驗室可以進去穿戴設定的人也會不相同，而且要顧及隱私，在上一批人的裝置設定好時，才能再讓別人進來。在進行實驗時，一旦進去穿戴就不能夠出來，直到結束才能出來，因為一但進行實驗就不能被終止，除非實驗完成，否則有可能對腦部造成損傷，因為裝置是互相影響的，所有的人設定完成並且實驗並計算完成是一個流程，也就是實驗結束，這時候大家才可以一起脫掉。

因為實驗進行招募制，人數有限，此實驗有一定的風險，所以人數很少
a 招募到 10 個人
b 招募到 5 個人
c 招募到 7 個人

他們發現會產生一種結果，當三間實驗尚未進行完成，外面剩下的人數無法滿足實驗室接下來要求的人數，造成沒有一間完成實驗，沒辦法出來，所以無法繼續進行。

實驗室 1 需要 3 個 a,1 個 b,2 個 c
目前已經有 0 個 a,1 個 b,2 個 c 在進行

實驗室 2 需要 7 個 a,2 個 b,2 個 c
目前已經有 4 個 a,0 個 b,1 個 c 在進行

實驗室 3 需要 9 個 a,3 個 b,2 個 c
目前已經有 3 個 a,2 個 b,2 個 c 在進行

一開始可用人數 a=3,b=2,c=2

## 二、Deadlock detection

### 1. 程式流程

每個 thread 在要求資源時的 while 程式碼，只要要求人數小於可用人數，可以進去就讓他進去進行實驗。

```
do
  {
      if(needa1 ==0) reqa1 = 0;                    //隨機產生要求人數的數量
      else reqa1 = (rand() % needa1) +1;
      if(needb1 ==0) reqb1 = 0;
      else reqb1 = (rand() % needb1) +1;
      if(needc1 ==0) reqc1 = 0;
      else reqc1 = (rand() % needc1) +1;

      printf("實驗室 1 要求:a=%d b=%d c=%d\n",reqa1,reqb1,reqc1);
      printf("實驗室 1 要求時可用人數剩餘:a=%d b=%d c=%d\n",a,b,c);
      while(temp==1);     //互斥存取
      temp=1;
      if((reqa1 <=a) && (reqb1<=b) &&(reqc1<=c))  //如果要求小於可用人數
      {
        printf("實驗室 1 要求通過\n");
        needa1 -= reqa1;
        needb1 -= reqb1;
        needc1 -= reqc1;
        geta1 += reqa1;
        getb1 += reqb1;
        getc1 += reqc1;
        a -= reqa1;
        b -= reqb1;
        c -= reqc1;
        temp =0;
      }
      else
      {
        printf("人數不夠!實驗室 1 的要求無法通過!\n");
        temp=0;
      }
      if(needa1==0 && needb1==0 && needc1==0) //做完了，把人數還回去
```

```
        {
          a += geta1;
          b += getb1;
          c += getc1;
          printf("實驗室 1 完成，返還人數，人數剩餘:a=%d b=%d
                  c=%d\n",a,b,c);
        }
        sleep(1);
    }while(needa1!=0 || needb1!=0 || needc1!=0);
```

**2. Deadlock 分析**

| MAX | | | |
|---|---|---|---|
| a | 3 | 1 | 2 |
| b | 7 | 2 | 2 |
| c | 9 | 3 | 2 |

✧ 運行正常沒死結的狀態(其中一種狀態)

初始

>>

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 2 | 3 | 0 | 0 | 3 | 2 | 2 |
| B | 4 | 0 | 1 | 3 | 2 | 1 | | | |
| B | 3 | 2 | 2 | 6 | 1 | 0 | | | |

實驗室 1 要求 a=3,b=0,c=0 之後，完成，返回人數，Finish1=[True]

>>

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 3 | 1 | 2 | 0 | 0 | 0 | 3 | 3 | 4 |
| B | 4 | 0 | 1 | 3 | 2 | 1 | | | |
| B | 3 | 2 | 2 | 6 | 1 | 0 | | | |

實驗室 2 要求 a=3,b=2,c=1 之後，返回人數，Finish2={True}

>>

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 3 | 1 | 2 | 0 | 0 | 0 | 7 | 3 | 5 |
| B | 6 | 2 | 3 | 0 | 0 | 0 | | | |
| B | 3 | 2 | 2 | 6 | 1 | 0 | | | |

剩下人數可滿足實驗室 3>>Finish3={True}

Finish[i]={True,True,True}

✧ 有死結的狀態(其中一種狀態)

初始

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 2 | 3 | 0 | 0 | 3 | 2 | 2 |
| B | 4 | 0 | 1 | 3 | 2 | 1 | | | |
| B | 3 | 2 | 2 | 6 | 1 | 0 | | | |

假設實驗室 1 要求 a=1,b=0,c=0 之後

>>

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| a | 1 | 1 | 2 | 2 | 0 | 0 | 2 | 2 | 2 |
| b | 4 | 0 | 1 | 3 | 2 | 1 | | | |
| b | 3 | 2 | 2 | 6 | 1 | 0 | | | |

實驗室 2 要求 a=1,b=1,c=1 之後

>>

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| a | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 1 |
| b | 5 | 1 | 2 | 2 | 1 | 0 | | | |
| b | 3 | 2 | 2 | 6 | 1 | 0 | | | |

實驗室 1 要求 a=1,b=0,c=0 之後

>>在這之後沒有一組能夠進入，因為 a 沒有了，又沒有實驗完成，死結

| | Allocation | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| a | 2 | 1 | 2 | 1 | 0 | 0 | 0 | 1 | 1 |
| b | 5 | 1 | 2 | 2 | 1 | 0 | | | |
| b | 3 | 2 | 2 | 6 | 1 | 0 | | | |

Finish[i] = {F,F,F}

### 3. 實驗步驟

使用 Dev C++撰寫 C 語言程式，並使用 pthread API 模擬三組實驗室

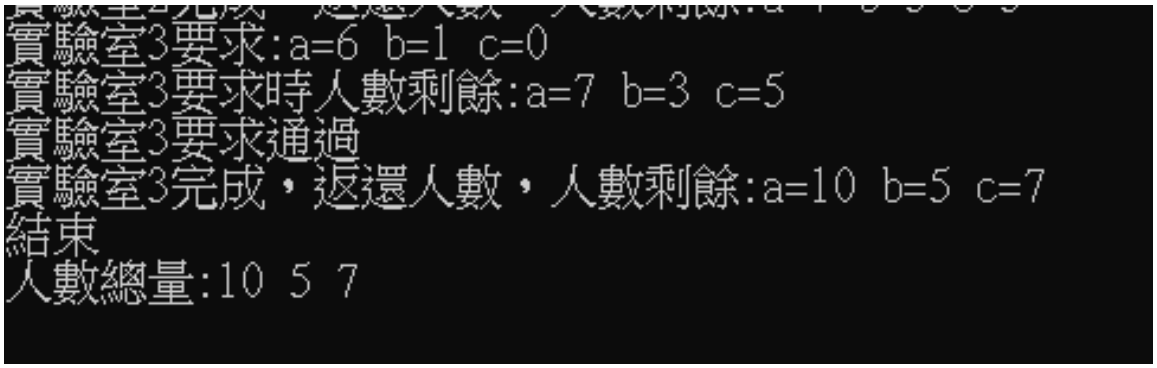
```
int a=3;     //可用的人數量
int b=2;     //
int c=2;     //

int maxa1=3,maxb1=1,maxc1=2;     //thread1最多要用的數量
int geta1=0,getb1=1,getc1=2;     //thread1目前手中人數的數量
int needa1,needb1,needc1;        //thread1還需要多少人數
int reqa1,reqb1,reqc1;           //thread1要求多少人數

int maxa2=7,maxb2=2,maxc2=2;     //thread2，承上
int geta2=4,getb2=0,getc2=1;
int needa2,needb2,needc2;
int reqa2,reqb2,reqc2;


int maxa3=9,maxb3=3,maxc3=2;     //thread3，承上
int geta3=3,getb3=2,getc3=2;
int needa3,needb3,needc3;
int reqa3,reqb3,reqc3;
```

宣告全域變數來記錄各 thread 狀況

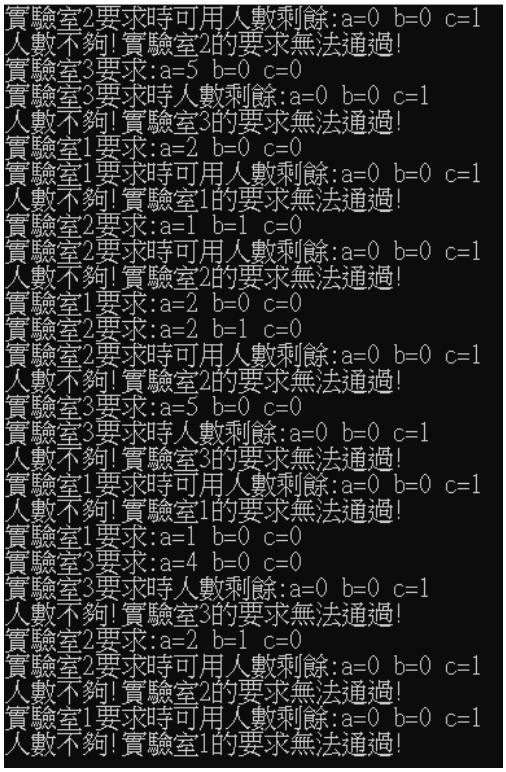
abc 為人數

get 為使用中人數

如果沒產生死結，則會回傳總資源數並結束



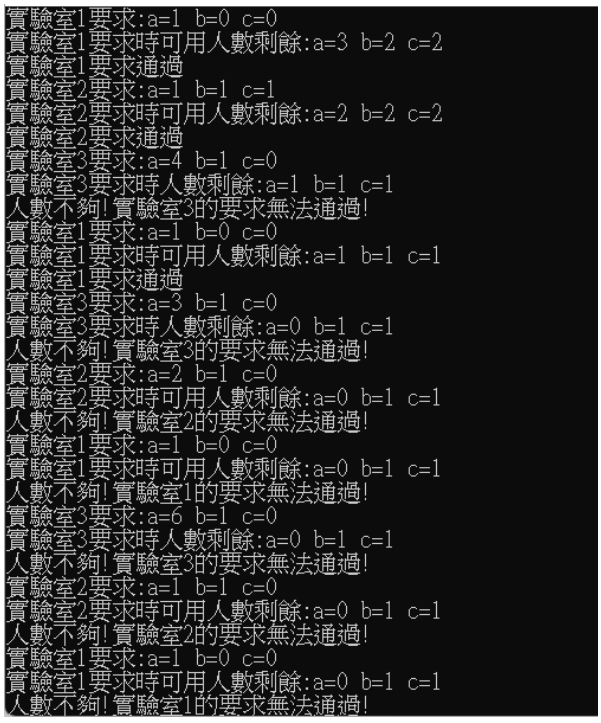
如果產生死結，則會一直要求但無法通過，一直循環，無法結束




## 4. 實驗結果與討論

Deadlock 分析的假設驗證

✧ 有死結的狀態(其中一種狀態)，一直要求但沒辦法通過，卡住



✧ 運行正常沒死結的狀態(其中一種狀態)

```
實驗室1要求:a=2 b=0 c=0
實驗室1要求時可用人數剩餘:a=3 b=2 c=2
實驗室1要求通過
實驗室3要求:a=5 b=1 c=0
實驗室3要求時人數剩餘:a=1 b=2 c=2
人數不夠!實驗室3的要求無法通過!
實驗室2要求:a=2 b=2 c=1
實驗室2要求時可用人數剩餘:a=1 b=2 c=2
人數不夠!實驗室2的要求無法通過!
實驗室3要求:a=3 b=1 c=0
實驗室3要求時人數剩餘:a=1 b=2 c=2
人數不夠!實驗室3的要求無法通過!
實驗室1要求:a=1 b=0 c=0
實驗室1要求時可用人數剩餘:a=1 b=2 c=2
實驗室1要求通過
實驗室1完成，返還人數，可用人數剩餘:a=3 b=3 c=4
實驗室2要求:a=2 b=2 c=1
實驗室2要求時可用人數剩餘:a=3 b=3 c=4
實驗室2要求通過
實驗室3要求:a=2 b=1 c=0
實驗室3要求時人數剩餘:a=1 b=1 c=3
人數不夠!實驗室3的要求無法通過!
實驗室2要求:a=1 b=0 c=0
實驗室2要求時可用人數剩餘:a=1 b=1 c=3
實驗室2要求通過
實驗室2完成，返還人數，人數剩餘:a=7 b=3 c=5
實驗室3要求:a=6 b=1 c=0
實驗室3要求時人數剩餘:a=7 b=3 c=5
實驗室3要求通過
實驗室3完成，返還人數，人數剩餘:a=10 b=5 c=7
結束
人數總量:10 5 7

--------------------------------
Process exited after 4.27 seconds with return value 0
請按任意鍵繼續 . . .
```

## 三、Deadlock 解決方法

### 1. 原理說明

當偵測到死結會發生時就不讓它通過，這次要求就不成立，透過使用 Banker Algorithm 來去判斷。

Banker Algorithm 我是用把每個判斷(可能會有的執行順序)都去判斷，這邊有三個 thread，所以會有 6 種順序，分別是 thread 123, 132, 213, 231, 312, 321。

### 2. 程式流程

原本程式碼只要要求人數小於能用人數就能過，這邊改使用 Banker Algorithm 判斷。

```
if(bankeralgorithm(1,0,0,geta1,getb1,getc1,geta2,getb2,getc2,geta3,getb3,getc3,needa1,needb1,needc1,needa2,needb2,needc2,needa3,needb3,needc3)==1)
{
    printf("child1 要求通過\n");
    needa1 -= reqa1;
    needb1 -= reqb1;
    needc1 -= reqc1;
    printf("needa1=%d needb1=%d needc1=%d\n",needa1,needb1,needc1);
    geta1 += reqa1;
    getb1 += reqb1;
    getc1 += reqc1;
    a -= reqa1;
    b -= reqb1;
    c -= reqc1;
    temp=0;
 }
```

---

Banker Algorithm 其中一條判斷式(123)

```
    //1
    if(need1<=testa && need1<=testb && need1<=testc)
    {
            testa += get1;
            testb += get2;
            testc += get3;
        //2>3
        if(need4<=testa && need5<=testb && need6<=testc)
```

```
{
        testa += get4;
        testb += get5;
        testc += get6;

    if(need7<=testa && need8<=testb && need9<=testc)
    {
            testa += get7;
            testb += get8;
            testc += get9;
            conti = 1;
    }
    else
    {
        testa -= get4;
        testb -= get5;
        testc -= get6;
    }
}
```

**3. 實驗步驟**

用 Banker Algorithm 判斷會不會產生死結

如果要求量大於可用人數，會直接回傳人數不夠

```
if(reqa1>testa || reqb1>testb || reqc1>testc)
{
    printf("人數不夠!");
    return 0;
}
```

如果假設的計算最後的人數是總人數，就是不會死結，可以通過，反之就是會產生死結，不給通過。

```
if(testa==10 && testb==5 && testc==7)
{
    return 1;
}
else
{
    printf("會產生死結!");
    return 0;
}
```

**4. 實驗結果與討論**

有死結的話會偵測出來，不能通過，必須重新要求，順利執行結束

```
child1要求:reqa1=3 reqb1=0 reqc1=0
child1要求時資源剩餘:a=3 b=2 c=2
child1要求通過

child1完成，返還資源，資源剩餘:a=3 b=3 c=4

main要求:reqa3=3 reqb3=1 reqc3=0
main要求時資源剩餘:a=3 b=3 c=4
會產生死結!main的要求無法通過!

child2要求:reqa2=3 reqb2=2 reqc2=1
child2要求時資源剩餘:a=3 b=3 c=4
child2要求通過

child2完成，返還資源，資源剩餘:a=7 b=3 c=5

main要求:reqa3=4 reqb3=1 reqc3=0
main要求時資源剩餘:a=7 b=3 c=5
main要求通過

main要求:reqa3=1 reqb3=0 reqc3=0
main要求時資源剩餘:a=3 b=2 c=5
main要求通過

main要求:reqa3=1 reqb3=0 reqc3=0
main要求時資源剩餘:a=2 b=2 c=5
main要求通過

main完成，返還資源，資源剩餘:a=10 b=5 c=7

結束
資源總量:10 5 7
```

## 四、心得感想（至少 200 字）

透過這一次的期末作業學到了不少經驗，包括 pthread 的使用以及 Banker Algorithm 的概念，實作上有點困難花了不少時間，在找 bug 上因為程式架構不夠清晰，且有點雜亂，困難不少，藉此讓我發現了我寫程式上面的排版與優化還可以有很大進步的空間，並且在過程中發現了許多在我們使用 Banker Algorithm 上與程式實現上的不同，我是設定成如果 Need 小於等於 Available 表示可以做完，就把 Allocation 加上去，但是在 thread 完成時，因為完成了，不需要加了，但程式架構上卻照樣會把 Allocation 加進去造成錯誤，這方面花了我不少時間想出錯誤的地方，以此為經驗，在往後的路上，能夠讓我更加警惕與要求。

## 五、參考文獻

C 語言 pthread 多執行緒平行化程式設計入門教學與範例
https://blog.gtwang.org/programming/pthread-multithreading-programming-in-c-tutorial/

## 六、附錄：程式原始碼

✧ 無使用 Banker Algorithm 程式碼，有死結

```
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>

int a=3; //可用的人數量
int b=2; //
int c=2; //

int maxa1=3, maxb1=1, maxc1=2;      //thread1 最多要用的數量
int geta1=0, getb1=1, getc1=2;      //thread1 目前手中人數的數量
int needa1, needb1, needc1;         //thread1 還需要多少人數
int reqa1, reqb1, reqc1;            //thread1 要求多少人數

int maxa2=7, maxb2=2, maxc2=2;      //thread2，承上
int geta2=4, getb2=0, getc2=1;
int needa2, needb2, needc2;
int reqa2, reqb2, reqc2;


int maxa3=9, maxb3=3, maxc3=2;      //thread3，承上
int geta3=3, getb3=2, getc3=2;
int needa3, needb3, needc3;
int reqa3, reqb3, reqc3;

int temp=0;

//thread1，child1
void *child1()
{
    srand((unsigned)time(NULL));   //時間亂數
    needa1 = maxa1-geta1;          //計算 need
    needb1 = maxb1-getb1;
    needc1 = maxc1-getc1;
```

```
do
{
    if(needa1 ==0) reqa1 = 0;                    //隨機產生要求人數的數量
    else reqa1 = (rand() % needa1) +1;
    if(needb1 ==0) reqb1 = 0;
    else reqb1 = (rand() % needb1) +1;
    if(needc1 ==0) reqc1 = 0;
    else reqc1 = (rand() % needc1) +1;
    printf("實驗室 1 要求:a=%d b=%d c=%d\n", reqa1, reqb1, reqc1);
    printf("實驗室 1 要求時可用人數剩餘:a=%d b=%d c=%d\n", a, b, c);
    while(temp==1);
    temp=1;
    if((reqa1 <=a) && (reqb1<=b) &&(reqc1<=c))   //如果要求沒超過剩餘
                                                   人數量，就可以通過去
                                                   使用
    {
      printf("實驗室 1 要求通過\n");
      needa1 -= reqa1; //因為通過了，給人數，這個 thread 的需求量，就
                         減上要求的人數量
      needb1 -= reqb1;
      needc1 -= reqc1;
      geta1 += reqa1;        //因為通過了，給人數，這個 thread 手上的人
                               數量，就加上要求的人數量
      getb1 += reqb1;
      getc1 += reqc1;
      a -= reqa1;            //人數減掉給的要求量
      b -= reqb1;
      c -= reqc1;
      temp=0;
    }
    else
    {
      printf("人數不夠!實驗室 1 的要求無法通過!\n");
      temp=0;
    }
    if(needa1==0 && needb1==0 && needc1==0) //如果個人數需要量都是
```

```
                                                    0，表示做完了，把人數還回去
        {
          a += geta1;
          b += getb1;
          c += getc1;
          printf("實驗室 1 完成，返還人數，可用人數剩餘:a=%d b=%d
                  c=%d\n", a, b, c);
        }
        sleep(1);
   }while(needa1!=0 || needb1!=0 || needc1!=0);     //還有需要人數，還沒
                                                          做完，繼續

   pthread_exit(NULL);   //thread 結束
}

//thread2，child2，跟 thread1 一樣
void *child2()
{
   srand((unsigned)time(NULL));
   needa2 = maxa2-geta2;
   needb2 = maxb2-getb2;
   needc2 = maxc2-getc2;


   do
   {
        if(needa2 ==0) reqa2 = 0;
        else reqa2 = rand() % needa2 +1;
        if(needb2 ==0) reqb2 = 0;
        else reqb2 = rand() % needb2 +1;
        if(needc2 ==0) reqc2 = 0;
        else reqc2 = rand() % needc2 +1;
        printf("實驗室 2 要求:a=%d b=%d c=%d\n", reqa2, reqb2, reqc2);
        printf("實驗室 2 要求時可用人數剩餘:a=%d b=%d c=%d\n", a, b, c);
        while(temp==1);
        temp=1;
        if((reqa2 <=a) && (reqb2<=b) &&(reqc2<=c))
        {
```

```
            printf("實驗室 2 要求通過\n");
            needa2 -= reqa2;
            needb2 -= reqb2;
            needc2 -= reqc2;
            geta2 += reqa2;
            getb2 += reqb2;
            getc2 += reqc2;
            a -= reqa2;
            b -= reqb2;
            c -= reqc2;
            temp=0;
          }
          else
          {
            printf("人數不夠!實驗室 2 的要求無法通過!\n");
            temp=0;
          }
          if(needa2==0 && needb2==0 && needc2==0)
          {
            a += geta2;
            b += getb2;
            c += getc2;
            printf("實驗室 2 完成，返還人數，人數剩餘:a=%d b=%d
                    c=%d\n", a, b, c);
          }
          sleep(1);
    }while(needa2!=0 || needb2!=0 || needc2!=0);

    pthread_exit(NULL);
}


//主程式，thread3
int main()
{

    srand((unsigned)time(NULL));
```

```
    pthread_t P1; //宣告一個 thread id 為 P1
    pthread_t P2;
    pthread_create(&P1, NULL, child1, ""); //(ID, NULL, thread 函式中自己
                                            取的名子, 傳入值)
    pthread_create(&P2, NULL, child2, "");

    needa3 = maxa3-geta3;
    needb3 = maxb3-getb3;
    needc3 = maxc3-getc3;


do
{
    if(needa3 ==0) reqa3 = 0;
    else reqa3 = rand() % needa3 +1;
    if(needb3 ==0) reqb3 = 0;
    else reqb3 = rand() % needb3 +1;
    if(needc3 ==0) reqc3 = 0;
    else reqc3 = rand() % needc3 +1;
    printf("實驗室 3 要求:a=%d b=%d c=%d\n", reqa3, reqb3, reqc3);
    printf("實驗室 3 要求時人數剩餘:a=%d b=%d c=%d\n", a, b, c);
    while(temp==1);
    temp=1;
    if((reqa3 <=a) && (reqb3<=b) &&(reqc3<=c))
    {
      printf("實驗室 3 要求通過\n");
      needa3 -= reqa3;
      needb3 -= reqb3;
      needc3 -= reqc3;
      geta3 += reqa3;
      getb3 += reqb3;
      getc3 += reqc3;
      a -= reqa3;
      b -= reqb3;
      c -= reqc3;
      temp=0;
    }
    else
```

```
        {
          printf("人數不夠!實驗室 3 的要求無法通過!\n");
          temp=0;
        }

        if(needa3==0 && needb3==0 && needc3==0)
        {
          a += geta3;
          b += getb3;
          c += getc3;
          printf("實驗室 3 完成，返還人數，人數剩餘:a=%d b=%d
                  c=%d\n", a, b, c);
        }
        sleep(1);
    }while(needa3!=0 || needb3!=0 || needc3!=0);


    pthread_join(P1, NULL); //如 thread 還沒結束，在這等待
    pthread_join(P2, NULL);
    printf("結束\n");
    printf("人數總量:%d %d %d\n", a, b, c);
    return 0;
}
```

✧ 有使用 Banker Algorithm 程式碼，無死結

```
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>

int a=3;
int b=2;
int c=2;
int temp=0;          //用來互斥 thread 使用 banker algorithm
int finish1=0, finish2=0, finish3=0;          //如=1，表示 thread 做完了

int maxa1=3, maxb1=1, maxc1=2;
int geta1=0, getb1=1, getc1=2;
int needa1=3, needb1=0, needc1=0;
int reqa1, reqb1, reqc1;

int maxa2=7, maxb2=2, maxc2=2;
int geta2=4, getb2=0, getc2=1;
int needa2=3, needb2=2, needc2=1;
int reqa2, reqb2, reqc2;

int maxa3=9, maxb3=3, maxc3=2;
int geta3=3, getb3=2, getc3=2;
int needa3=6, needb3=1, needc3=0;
int reqa3, reqb3, reqc3;

/* bank algorithm
按照 3 個 thread，六種可能一一執行測試
1>2>3, 1>3>2, 2>1>3, 2>3>1, 3>1>2, 3>2>1
*/
int bankeralgorithm(int flag1, int flag2, int flag3, int get1, int get2, int get3, int
get4, int get5, int get6, int get7, int get8, int get9, int need1, int need2, int
need3, int need4, int need5, int need6, int need7, int need8, int need9)
{
    temp=1;
    int testa=a, testb=b, testc=c;
    int conti=0;  // conti 為 1，就不會繼續判斷其他選擇
```

```
if(flag1==1) //flag1 為 thread1 進入要求人數
{
    if(reqa1>testa || reqb1>testb || reqc1>testc)
    {
        printf("人數不夠!");
        return 0;
    }
    else
    {
        if(need4==0 &&need5==0 &&need6==0)
        {
            get4=0;
            get5=0;
            get6=0;
        }
        if(need7==0 &&need8==0 &&need9==0)
        {
            get7=0;
            get8=0;
            get9=0;
        }
        if(need1==0 &&need2==0 &&need3==0)
        {
            get1=0;
            get2=0;
            get3=0;
        }
        else
        {
            get1 += reqa1;
            get2 += reqb1;
            get3 += reqc1;
        }

        need1 -= reqa1;
        need2 -= reqb1;
        need3 -= reqc1;
```

```
            testa -= reqa1;
            testb -= reqb1;
            testc -= reqc1;
        }
    }
    else if(flag2==1)
    {
        if(reqa2>testa || reqb2>testb || reqc2>testc)
        {
            printf("人數不夠!");
            return 0;
        }
        else
        {
            if(need1==0 &&need2==0 &&need3==0)
            {
                get1=0;
                get2=0;
                get3=0;
            }
            if(need7==0 &&need8==0 &&need9==0)
            {
                get7=0;
                get8=0;
                get9=0;
            }
            if(need4==0 &&need5==0 &&need6==0)
            {
                get4=0;
                get5=0;
                get6=0;
            }
            else
            {
                get4 += reqa2;
                get5 += reqb2;
                get6 += reqc2;
```

```
            }

            need4 -= reqa2;
            need5 -= reqb2;
            need6 -= reqc2;

            testa -= reqa2;
            testb -= reqb2;
            testc -= reqc2;
        }
    }
    else if(flag3==1)
    {
        if(reqa3>testa || reqb3>testb || reqc3>testc)
        {
            printf("人數不夠!");
            return 0;
        }
        else
        {
            if(need1==0 &&need2==0 &&need3==0)
            {
                get1=0;
                get2=0;
                get3=0;
            }
            if(need4==0 &&need5==0 &&need6==0)
            {
                get4=0;
                get5=0;
                get6=0;
            }

            if(need7==0 &&need8==0 &&need9==0)
            {
                get7=0;
                get8=0;
                get9=0;
```

```
                }
                else
                {
                    get7 += reqa3;
                    get8 += reqb3;
                    get9 += reqc3;
                }
                need7 -= reqa3;
                need8 -= reqb3;
                need9 -= reqc3;

                testa -= reqa3;
                testb -= reqb3;
                testc -= reqc3;
            }
        }



//開始判斷
//判斷從 thread1 開始所有可能，1>2>3, 1>3>2

        //1
        if(need1<=testa && need1<=testb && need1<=testc)
        {
                testa += get1;
                testb += get2;
                testc += get3;
            //2>3
            if(need4<=testa && need5<=testb && need6<=testc)
            {
                    testa += get4;
                    testb += get5;
                    testc += get6;

                if(need7<=testa && need8<=testb && need9<=testc)
                {
                        testa += get7;
```

```
                testb += get8;
                testc += get9;
                conti = 1;
        }
        else
        {
            testa -= get4;
            testb -= get5;
            testc -= get6;
        }
}

//3>2
if(need7<=testa && need8<=testb && need9<=testc &&
    conti==0)
{
            testa += geta3;
            testb += getb3;
            testc += getc3;

        if(need4<=testa && need5<=testb && need6<=testc)
        {
                testa += get4;
                testb += get5;
                testc += get6;
                conti = 1;
        }
        else
        {
            testa -= geta3;
            testb -= getb3;
            testc -= getc3;
        }
}
if(conti==0)
{
        testa -= get1;
        testb -= get2;
```

```
            testc -= get3;
        }

}

//如果 thread1 沒有順序，第 2 個 thread 開始判斷，2>1>3, 2>3>1
//2
if(need4<=testa && need5<=testb && need6<=testc && conti==0)
{

            testa += get4;
            testb += get5;
            testc += get6;
        //1>3
        if(need1<=testa && need2<=testb && need3<=testc)
        {

                    testa += get1;
                    testb += get2;
                    testc += get3;

                    if(need7<=testa && need8<=testb && need9<=testc)
                    {
                                testa += get7;
                                testb += get8;
                                testc += get9;
                                conti = 1;
                    }
                    else
                    {
                         testa -= get1;
                         testb -= get2;
                         testc -= get3;
                    }
        }

        //3>1
        if(need7<=testa && need8<=testb && need9<=testc &&
```

```
        conti==0)
    {
            testa += get7;
            testb += get8;
            testc += get9;
        if(need1<=testa && need2<=testb && need3<=testc)
        {
                testa += get1;
                testb += get2;
                testc += get3;
                conti = 1;
        }
        else
        {
            testa -= get7;
            testb -= get8;
            testc -= get9;
        }
    }
    if(conti==0)
    {
        testa -= get4;
        testb -= get5;
        testc -= get6;
    }
}
```

```
//如果 thread1 與 thread2 沒有順序，第 3 個 thread 開始判斷，
  3>1>2, 3>2>1
//3
if(need7<=testa && need8<=testb && need9<=testc && conti==0)
{
        testa += get7;
        testb += get8;
        testc += get9;
```

```
//1>2
if(need1<=testa && need2<=testb && need3<=testc)
{
        testa += get1;
        testb += get2;
        testc += get3;

        if(need4<=testa && need5<=testb && need6<=testc)
        {
                testa += get4;
                testb += get5;
                testc += get6;
                conti = 1;
        }
        else
        {
             testa -= get1;
             testb -= get2;
             testc -= get3;
        }
}

//2>1
if(need4<=testa && need5<=testb && need6<=testc &&
   conti==0)
{
        testa += get4;
        testb += get5;
        testc += get6;

     if(need1<=testa && need2<=testb && need3<=testc)
     {
             testa += get1;
             testb += get2;
             testc += get3;
             conti = 1;
     }
```

```
                else
                {
                    testa -= get4;
                    testb -= get5;
                    testc -= get6;
                }
            }
            if(conti==0)
            {
                testa -= get7;
                testb -= get8;
                testc -= get9;
            }
        }

        if(testa==10 && testb==5 && testc==7) //如果最後回去的人數
                                              等於總人數，表示有順序假設成立
        {
            return 1;
        }
        else
        {
            printf("會產生死結!");
            return 0;
        }
}



//therad1
void *child1()
{
    srand((unsigned)time(NULL));
    needa1 = maxa1-geta1;
    needb1 = maxb1-getb1;
    needc1 = maxc1-getc1;
```

```
do
{
    if(needa1 ==0) reqa1 = 0;
    else reqa1 = (rand() % needa1) +1;
    if(needb1 ==0) reqb1 = 0;
    else reqb1 = (rand() % needb1) +1;
    if(needc1 ==0) reqc1 = 0;
    else reqc1 = (rand() % needc1) +1;
    printf("child1 要求:reqa1=%d reqb1=%d
            reqc1=%d\n", reqa1, reqb1, reqc1);
    printf("child1 要求時人數剩餘:a=%d b=%d c=%d\n", a, b, c);
    while(temp==1);
    temp=1;

if(bankeralgorithm(1, 0, 0, geta1, getb1, getc1, geta2, getb2, getc2, geta3, getb3, getc
3, needa1, needb1, needc1, needa2, needb2, needc2, needa3, needb3, needc3)==1)

    {
      printf("child1 要求通過\n\n");
      needa1 -= reqa1;
      needb1 -= reqb1;
      needc1 -= reqc1;
      geta1 += reqa1;
      getb1 += reqb1;
      getc1 += reqc1;
      a -= reqa1;
      b -= reqb1;
      c -= reqc1;
      temp=0;
    }
    else
    {
      printf("child1 的要求無法通過!\n\n");
      temp = 0;
    }

    if(needa1==0 && needb1==0 && needc1==0)
    {
```

```
            a += geta1;
            b += getb1;
            c += getc1;
            printf("child1 完成，返還人數，人數剩餘:a=%d b=%d
                    c=%d\n\n", a, b, c);
            finish1 =1;
        }
        sleep(1);
    }while(needa1!=0 || needb1!=0 || needc1!=0);

    pthread_exit(NULL);
}

//thread2
void *child2()
{
    srand((unsigned)time(NULL));
    needa2 = maxa2-geta2;
    needb2 = maxb2-getb2;
    needc2 = maxc2-getc2;

    do
    {
        if(needa2 ==0) reqa2 = 0;
        else reqa2 = rand() % needa2 +1;
        if(needb2 ==0) reqb2 = 0;
        else reqb2 = rand() % needb2 +1;
        if(needc2 ==0) reqc2 = 0;
        else reqc2 = rand() % needc2 +1;
        printf("child2 要求:reqa2=%d reqb2=%d
                reqc2=%d\n", reqa2, reqb2, reqc2);
        printf("child2 要求時人數剩餘:a=%d b=%d c=%d\n", a, b, c);
        while(temp==1);
        temp=1;

if(bankeralgorithm(0, 1, 0, geta1, getb1, getc1, geta2, getb2, getc2, geta3, getb3, getc
3, needa1, needb1, needc1, needa2, needb2, needc2, needa3, needb3, needc3)==1)
        {
```

```
            printf("child2 要求通過\n\n");
            needa2 -= reqa2;
            needb2 -= reqb2;
            needc2 -= reqc2;
            geta2 += reqa2;
            getb2 += reqb2;
            getc2 += reqc2;
            a -= reqa2;
            b -= reqb2;
            c -= reqc2;
            temp=0;
          }
          else
          {
            printf("child2 的要求無法通過!\n\n");
            temp = 0;
          }
          if(needa2==0 && needb2==0 && needc2==0)
          {
            a += geta2;
            b += getb2;
            c += getc2;
            printf("child2 完成，返還人數，人數剩餘:a=%d b=%d
c=%d\n\n", a, b, c);
            finish2=1;
          }
          sleep(1);
    }while(needa2!=0 || needb2!=0 || needc2!=0);

    pthread_exit(NULL);
}


// thread3
int main()
{

    srand((unsigned)time(NULL));
```

```
    pthread_t P1;
    pthread_t P2;
    pthread_create(&P1, NULL, child1,"");
    pthread_create(&P2, NULL, child2,"");

    needa3 = maxa3-geta3;
    needb3 = maxb3-getb3;
    needc3 = maxc3-getc3;


  do
  {
      if(needa3 ==0) reqa3 = 0;
      else reqa3 = rand() % needa3 +1;
      if(needb3 ==0) reqb3 = 0;
      else reqb3 = rand() % needb3 +1;
      if(needc3 ==0) reqc3 = 0;
      else reqc3 = rand() % needc3 +1;
      printf("main 要求:reqa3=%d reqb3=%d
              reqc3=%d\n", reqa3, reqb3, reqc3);
      printf("main 要求時人數剩餘:a=%d b=%d c=%d\n", a, b, c);
      while(temp==1);
      temp=1;

if(bankeralgorithm(0, 0, 1, geta1, getb1, getc1, geta2, getb2, getc2, geta3, getb3, getc
3, needa1, needb1, needc1, needa2, needb2, needc2, needa3, needb3, needc3)==1)
      {
        printf("main 要求通過\n\n");
        needa3 -= reqa3;
        needb3 -= reqb3;
        needc3 -= reqc3;
        geta3 += reqa3;
        getb3 += reqb3;
        getc3 += reqc3;
        a -= reqa3;
        b -= reqb3;
        c -= reqc3;
```

```
            temp=0;
        }
        else
        {
            printf("main 的要求無法通過!\n\n");
            temp = 0;
        }

        if(needa3==0 && needb3==0 && needc3==0)
        {
            a += geta3;
            b += getb3;
            c += getc3;
            printf("main 完成，返還人數，人數剩餘:a=%d b=%d
                    c=%d\n\n", a, b, c);
            finish3=1;
        }
        sleep(1);
    }while(needa3!=0 || needb3!=0 || needc3!=0);


    pthread_join(P1, NULL);
    pthread_join(P2, NULL);
    printf("結束\n");
    printf("人數總量:%d %d %d\n", a, b, c);
    return 0;
}
```